

NAVAL POSTGRADUATE SCHOOL

Monterey, California

AD-A208 128



THESIS

AN ENGINEERING METHODOLOGY
FOR IMPLEMENTING AND
TESTING VLSI CIRCUITS

by

Walter F. Corliss II

March 1989

Thesis Advisor

Herschel H. Loomis, Jr.

Approved for public release; distribution is unlimited.

DTIC
ELECTE
MAY 26 1989
S H D
CB

Unclassified

security classification of this page

REPORT DOCUMENTATION PAGE

1a Report Security Classification Unclassified			1b Restrictive Markings		
2a Security Classification Authority			3 Distribution Availability of Report		
2b Declassification Downgrading Schedule			Approved for public release; distribution is unlimited.		
4 Performing Organization Report Number(s)			5 Monitoring Organization Report Number(s)		
6a Name of Performing Organization Naval Postgraduate School		6b Office Symbol (if applicable) 62	7a Name of Monitoring Organization Naval Postgraduate School		
6c Address (city, state, and ZIP code) Monterey, CA 93943-5000			7b Address (city, state, and ZIP code) Monterey, CA 93943-5000		
8a Name of Funding Sponsoring Organization		8b Office Symbol (if applicable)	9 Procurement Instrument Identification Number		
8c Address (city, state, and ZIP code)			10 Source of Funding Numbers		
			Program Element No	Project No	Task No
			Work Unit Accession No		
11 Title (include security classification) AN ENGINEERING METHODOLOGY FOR IMPLEMENTING AND TESTING VLSI CIRCUITS					
12 Personal Author(s) Walter F. Corliss II					
13a Type of Report Master's Thesis		13b Time Covered From To		14 Date of Report (year, month, day) March 1989	
				15 Page Count 121	
16 Supplementary Notation The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.					
17 Cosati Codes			18 Subject Terms (continue on reverse if necessary and identify by block number)		
Field	Group	Subgroup	VLSI, MAGIC, MOSSIM II, DAS9100, DVS50, Digital Test Facilities		
19 Abstract (continue on reverse if necessary and identify by block number)					
<p>The engineering methodology for producing a fully tested VLSI chip from a design layout is presented. A 16-bit correlator, NPS CORN88, that was previously designed, was used as a vehicle to demonstrate this methodology. The study of the design and simulation tools, MAGIC and MOSSIM II, was the focus of the design and validation process. The design was then implemented and the chip was fabricated by MOSIS.</p> <p>This fabricated chip was then used to develop a testing methodology for using the digital test facilities at NPS. NPS CORN88 was the first full custom VLSI chip, designed at NPS, to be tested with the NPS digital analysis system, Tektronix DAS 9100 series tester. The capabilities and limitations of these test facilities are examined within this thesis. NPS CORN88 test results are included to demonstrate the capabilities of the digital test system. A translator, MOS2DAS, was developed to convert the MOSSIM II simulation program to the input files required by the DAS 9100 device verification software, 91DVS. Finally, a tutorial for using the digital test facilities, including the DAS 9100 and associated support equipments, is included as an appendix.</p>					
20 Distribution Availability of Abstract			21 Abstract Security Classification		
<input checked="" type="checkbox"/> unclassified unlimited <input type="checkbox"/> same as report <input type="checkbox"/> DTIC users			Unclassified		
22a Name of Responsible Individual Herschel H. Loomis, Jr.			22b Telephone (include Area code) (408) 646-3214		22c Office Symbol 621 m

DD FORM 1473.84 MAR

82 APR edition may be used until exhausted
All other editions are obsolete

security classification of this page

Unclassified

Approved for public release; distribution is unlimited.

AN ENGINEERING METHODOLOGY
FOR IMPLEMENTING AND
TESTING VLSI CIRCUITS

by

Walter F. Corliss II
Lieutenant Commander, United States Navy
B.S., Purdue University, 1976

Submitted in partial fulfillment of the
requirements for the degree of

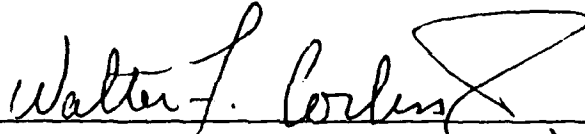
MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL

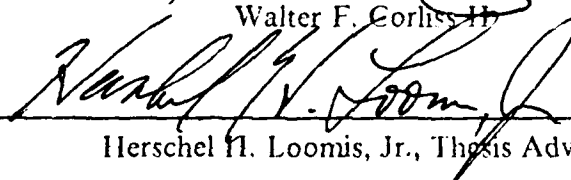
March 1989

Author:

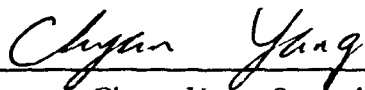


Walter F. Corliss II

Approved by:



Herschel H. Loomis, Jr., Thesis Advisor



Chyan Yang, Second Reader



John P. Powers, Chairman,
Department of Electrical and Computer Engineering



Gordon E. Schacher,
Dean of Science and Engineering

ABSTRACT

The engineering methodology for producing a fully tested VLSI chip from a design layout is presented. A 16-bit correlator, NPS CORN88, that was previously designed, was used as a vehicle to demonstrate this methodology. The study of the design and simulation tools, MAGIC and MOSSIM II, was the focus of the design and validation process. The design was then implemented and the chip was fabricated by MOSIS.

This fabricated chip was then used to develop a testing methodology for using the digital test facilities at NPS. NPS CORN88 was the first full custom VLSI chip, designed at NPS, to be tested with the NPS digital analysis system, Tektronix DAS 9100 series tester. The capabilities and limitations of these test facilities are examined within this thesis. NPS CORN88 test results are included to demonstrate the capabilities of the digital test system. A translator, MOS2DAS, was developed to convert the MOSSIM II simulation program to the input files required by the DAS 9100 device verification software, 91DVS. Finally, a tutorial for using the digital test facilities, including the DAS 9100 and associated support equipments, is included as an appendix.



Accession For	
NTIS GRA&I	<input checked="" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By _____	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

I. INTRODUCTION	1
A. BACKGROUND	1
B. THESIS GOALS	5
C. THESIS ORGANIZATION	5
II. DESIGN IMPLEMENTATION	6
A. NPS CORN88	6
1. Initial Design Methodology	6
2. Functional Description	8
3. Partitioning for Testability	12
B. MAGIC LAYOUT SYSTEM	13
1. Silicon Semiconductor Technology Overview	13
2. MAGIC Layout Features	17
a. Invoking Commands	17
b. Basic Magic Tools	18
3. MAGIC External Interfaces	19
a. Circuit Extraction	19
b. Caltech Intermediate Form (CIF) / Calma Stream Format	19
4. Design-Rule Checking	20
a. Single Cell Layouts	20
b. Hierarchical Layout Rules	22
5. MAGIC Layout of NPS CORN88	23
C. SIMULATION	24
1. Preparation for Simulation	24
2. Background	24
3. Overview of Capabilities	25
4. NPS CORN88 Test Vectors	28
D. CFL	28
1. Hierarchical Assembly of Magic Leaf Cells	28
2. CFL Advantages Disadvantages	30
E. MOSIS	31

1. Background	31
2. Preparations Prior to Fabrication	33
3. MOSIS Quality Control	33
III. NPS VLSI DIGITAL TEST FACILITIES	35
A. BACKGROUND	35
B. DIGITAL ANALYSIS SYSTEM (DAS) 9100	35
1. Overview	35
2. Data Acquisition	36
a. Modules	36
b. Menus	37
3. DAS 9100 Pattern Generation.	38
4. External Interfaces	39
C. 9100 DEVICE VERIFICATION SOFTWARE	40
1. Background	40
2. DVS50 Capabilities and Limitations	41
3. DVS50 Input Programs	41
4. DVS50 Menus	45
5. NPS CORN88	47
a. Set-up	47
b. Results	49
IV. MOS2DAS TRANSLATOR	58
A. BACKGROUND	58
B. MOS2DAS CAPABILITIES AND LIMITATIONS	58
1. MOSSIM II Commands	58
2. File Conversion Process	60
C. OPERATIONAL SHELL	65
V. CONCLUSIONS	69
A. SUMMARY	69
B. RECOMMENDATIONS	70
C. BENEFITS	71
APPENDIX A. NPS DIGITAL TEST FACILITY TUTORIAL	72

A. PURPOSE	72
B. CAPABILITIES AND LIMITATIONS	72
C. DVSS0 INPUT FILES	73
D. TEST PROGRAM COMPILATION	76
E. DUT SET UP	78
F. RUNNING THE TEST	80
G. CONCLUSIONS	82
APPENDIX B. MOSTRANS TRANSLATION PROGRAM	89
A. TRANSLATOR SHELL PROGRAM	89
B. TRANSLATOR SOURCE PROGRAM	91
APPENDIX C. VECTOR GENERATION PROGRAMS	103
A. VECTOR SHELL PROGRAM	103
B. VECTOR GENERATION C SOURCE CODE	104
LIST OF REFERENCES	106
INITIAL DISTRIBUTION LIST	109

LIST OF TABLES

Table 1.	NPS CORN88 INPUT PINS	10
Table 2.	NPS CORN88 OUTPUT PINS	10
Table 3.	CORRELATOR RESULTS (NO MASKING)	11
Table 4.	CORRELATOR RESULTS (WITH MASKING)	11
Table 5.	ABBREVIATIONS FOR SETS OF TILE TYPES	21
Table 6.	DESIGN-RULE WIDTH REQUIREMENTS (IN LAMBDA UNITS) .	21
Table 7.	DESIGN-RULE WIDTH REQUIREMENTS (IN LAMBDA UNITS) .	22
Table 8.	DIGITAL ANALYSIS SYSTEM (DAS) CAPABILITIES	36
Table 9.	DVS50 MODULE SUPPORT	41
Table 10.	NPS CORN88 INITIAL REGISTER SET-UP	52
Table 11.	NPS CORN88 REFERENCE AND MASKING INITIAL REGISTER VALUES	56

LIST OF FIGURES

Figure 1.	Design Implementation Process	7
Figure 2.	NPS CORN88 Block Diagram	9
Figure 3.	MAGIC Materials Palette	14
Figure 4.	MAGIC Layout of a CMOS Inverter	15
Figure 5.	Manhattan Distance	21
Figure 6.	NPS CORN88 Hierarchical Structure	23
Figure 7.	MAGIC to Simulation File Conversion Process	25
Figure 8.	MAGIC ".mag" File to ".ntk" file Commands	26
Figure 9.	NPS CORN88 MOSSIM II Source Program	27
Figure 10.	CFL Composite Chip Assembly	29
Figure 11.	NPS CORN88 CFL Block Structure	31
Figure 12.	CFL CR_regcell	32
Figure 13.	NPS CORN88 DVS50 .SRC File	43
Figure 14.	NPS CORN88 DVS50 .DAS File	44
Figure 15.	DVS50 Processing Flow	46
Figure 16.	NPS CORN88 Channel Specifications	48
Figure 17.	NPS Test Jig	50
Figure 18.	NPS CORN88 Data Register Test Results	51
Figure 19.	NPS CORN88 Combiner Adder Test Results	53
Figure 20.	NPS CORN88 Reference Register Test Results	54
Figure 21.	NPS CORN88 Masking Test Results	55
Figure 22.	NPS CORN88 Serial Input Test Results	57
Figure 23.	NPS CORN88 MOSSIM II Simulation Program	61
Figure 24.	MOS2DAS ".das" File	62
Figure 25.	MOS2DAS ".sim" File	64
Figure 26.	DAS 9100 and MOSSIM Sampling Differences	65
Figure 27.	MOS2DAS ".src" File	66
Figure 28.	MOS2DAS C-Shell Operating Program	67
Figure 29.	ALU181 Specifications	74
Figure 30.	DOS Commands and File Directory Flow	75
Figure 31.	ALU181 ".src" File	77

Figure 32. ALU181 ".das" File	78
Figure 33. ALU181 ".sim" File	79
Figure 34. DVS50 Main Menu	81
Figure 35. ALU181 Channel Specifications	83
Figure 36. DVS50 Test Menu	84
Figure 37. Test Jig	85
Figure 38. Physical Location of the Modules	86
Figure 39. ALU181 Results	87
Figure 40. Display Help Menu	88

ACKNOWLEDGEMENT

The assistance provided by the staff of the Computer Lab in the Electrical and Computer Engineering Department at the Naval Postgraduate School was invaluable to the completion of this research. Thank you Dave, Bob, and Elaine for your professional assistance, your patience, and your sense of humor.

I. INTRODUCTION

A. BACKGROUND

"For the past 20 years chip complexity and thus the functional capability of a single integrated circuit (IC) have roughly doubled every year." [Ref. 1, p. 1]. Chip development has progressed from the small scale integration (SSI) to that of the very large scale integrated (VLSI) circuit. Current technology considers a VLSI chip to contain more than 100,000 gates and transistors. As advances have been made in the manufacturing process and composition materials, the chip density has become greater and the chips operate at a faster rate. This growth in more devices per unit area and complexity of VLSI circuits has required that the testing and validation process of these chips become more sophisticated. Most custom VLSI chips that are developed are produced for a specific purpose and are referred to as application specific integrated circuits (ASICs). For each of these full custom chips a methodology must be used to implement and validate the design, functionally test the completed cell before manufacture, and comprehensive testing of the VLSI chip after it returns from the manufacture's foundry. Again, due to the complexity of the circuits this methodology is often driven by the computer tools that are available to assist the circuit designer and test engineer. Computer-aided design (CAD) computer-aided testing (CAT) and computer simulation routines have become essential in the development of high performance VLSI chips.

CAD programs assist the VLSI chip designer in the physical layout of the design circuitry. These CAD programs help isolate the designer from the masking details, allowing the design effort to concentrate on higher level development. Illustrating the explosion of interest in the area of CAD development, a recent *Computer Design* issue [Ref. 2], listed over 29 commercial vendors with 53 different models of printed circuit board CAD systems. The printed circuit board based CAD system uses a combination of specially designed hardware and software to layout the design. At the Naval Postgraduate School (NPS), the CAD layout tools of MAGIC and GENESIL are implemented in software.

MAGIC, developed at the University of California at Berkeley, is a research software tool that is installed on the Digital Vax 780, 1185 and three Integrated Solutions Incorporated (ISI) work stations. It provides the circuit designer with an interactive layout editing system for large-scale MOS custom integrated circuits and is discussed in

detail in [Ref. 3]. It allows the designer to engage in full custom VLSI design from the transistor to the library cell level. Knowledge of silicon semiconductor technology is required for various forms of transistor development but designs at the gate level and higher are possible by using library cells that have already been designed. Once the cells have been developed, or recalled from the cell library, MAGIC can assemble them hierarchically into the required circuitry. It also provides a powerful router to establish the connectivity between cells. One of the key advantages of using MAGIC is the feature that is provided called design-rule checking. MAGIC maintains a continuous design-rule check to ensure that the proper topology and layout parameters are maintained for the technology (CMOS, NMOS, PMOS) as well as the technology required by the foundry. This design-rule check is updated incrementally during the design and if an error occurs then that area of concern is highlighted by white dots. Magic does not offer any timing or fault isolation simulation, but the MAGIC design file can be extracted and several simulation tools can then be run to validate the design.

"The GENESIL Silicon Development System is a comprehensive turnkey design automation system for creation of integrated circuits, NMOS or CMOS." [Ref. 4]. GENESIL is a commercial CAD tool from Silicon Compiler Systems that also runs on the VAX with an interface either a schematic capture graphics display terminal or from its own graphics terminal. The user manual for the system [Ref. 4] stresses that the goal of GENESIL is to make IC design approachable to systems designers who lack IC designer expertise. To that end GENESIL is a menu driven interactive layout editing system that concentrates on high-level systems design. There are hundreds of complex functional parts available in its library of cells, and the designer selects the cells from which to build and connects them together with the netlist of routing commands.

It is important to stress the difference between this method of assembling cells in GENESIL contrasted to that of the MAGIC CAD tools. The GENESIL system designer does not view or build the library cells below the gate level. No knowledge of silicon semiconductor technology is required by the designer and there is no reason for GENESIL to provide for design-rule checking because the library cells have already met all design performance criteria. GENESIL also provides a design verification package as part of its software package. Timing analysis, power requirement analysis, and automatic test generation allow the designer to functionally verify that the chip design is sound. This is done entirely within the GENESIL system without interfacing with other simulation software. This definitely speeds up the design and verification process of the custom VLSI chip considerably. Decreasing the development time of the VLSI chip en-

courages the designer to experiment and optimize the circuit design. Once the design is set and verified, GENESIL can translate the design to the geometric masking data used by several different manufacturing foundries. From design concept to fabrication tooling, GENESIL produces high performance custom ASIC designs. Three NPS graduate thesis illustrate the robust features of the GENESIL silicon compiler. Settle examined the design methodology of using the GENESIL silicon compiler by designing a pipeline 16-bit multiplier [Ref. 5]. Rocky implemented a Kalman Filter Algorithm as an ASIC on GENESIL [Ref. 6]. Davidson examined testability strategy using GENESIL's features on Automatic Test Generation (ATG) and the test latch library cell [Ref. 7]. Reviewing these thesis will provide a solid foundation for utilizing the GENESIL silicon compiler.

MAGIC is the CAD tool used by NPS students enrolled in VLSI course work and was used for the development of NPS CORN88 during this thesis research. It stresses silicon semiconductor technology and provides a greater insight into masking design and layout rules. But, verifying a MAGIC design requires that the circuit must be extracted into a format that a computer simulation model will utilize. The three electrical circuit simulation models that are widely used at NPS are MOSSIM II, SPICE, and Register Notational Language (RNL).

MOSSIM II is a logic simulator based on the switch-level logic model. It models a MOS digital circuit as a network of nodes connected by transistor switches and can accurately model such circuit structures as bi-directional pass transistors, ratioed and complementary logic buses, dynamic memory and charge sharing. MOSSIM II, described in detail in the user's manual [Ref. 8], is designed primarily for simulating clocked systems where a clocking scheme consists of a set of state sequences to be applied cyclically to a set of nodes. Identifying the input forcing vectors and the outputs to be watched, MOSSIM II will run the modeled circuit and produce the simulated output results at each clock cycle identified. Although MOSSIM II provides minimum timing information, it can identify potential timing errors. Because MOSSIM II is based on a switch-level model, it can handle large-scale sequential and combinational logic design projects relatively quickly, but it can not be used on analog designs.

Another digital circuit simulator, RNL, is a timing simulator for digital MOS circuits. It is an event-driven simulator that uses a simple resistance and capacitance (RC) model of the circuit to estimate node transistor time and to estimate the effects of charge sharing. RNL provides similar features to MOSSIM II which are outlined in the RNL user manual [Ref. 9]. The user interface for the RNL simulation program is a version

of LISP. RNL appears to run as fast as MOSSIM II but it can not handle bi-directional pass transistors. It also can only be used to simulate digital circuitry and can not be used on analog circuitry.

SPICE simulation is based on the numerical solution of the network's differential equations. After the MAGIC file is extracted, each transistor, resistor, and capacitor is modeled. Detailed instructions on using SPICE are included in users guide [Ref. 10] and are applicable for analog and digital circuits. It is important to note that because SPICE computes the differential equation for each transistor it is very slow and is practical for only small circuits or in validating library cells. It does produce superior simulation timing results, electrical parameters at all observed nodes, and statistical features that allow the designer to examine in detail the complete operation of the circuit. For that reason SPICE is valuable in validating individual leaf cells, but is seldom used on the entire VLSI digital designs.

Once the final design has been functionally validated it must be packaged for the manufacturing foundry. MOSIS is an acronym for MOS Implementation System and acts as a clearing house facility to handle the details in manufacturing a chip. Coordinated from the Information Sciences Institute of the University of Southern California MOSIS ensures that the chip has correct syntax upon receipt of the layout geometry and delivers a set of bonded and packaged chips to the user. This can be done at a very reasonable cost because several different designs of the same silicon technology share the same silicon wafer. The complete process from submitting the VLSI chip layout to the delivery by MOSIS of the fabricated chip is outlined in the users manual [Ref. 11].

Once the VLSI chip has been manufactured and returned to NPS, the Tektronix DAS 9100 series tester is used to test the fabricated chip. This stand-alone, digital analysis system can functionally verify that the VLSI chip meets the required performance standards for various operational speeds. The user manual for the Tektronix DAS 9100 series tester [Ref. 12] details the pattern generation and data acquisition features. The DAS 9100 tester at NPS can also link to a personal computer (PC) and device verification software (DVS). This configuration generates test patterns in vector format and configures the Tektronix DAS 9100 tester to the device under test (DUT).

When designing a custom ASIC, it is extremely important for the design engineer to take into consideration and understand all the tools that are available. The testing and testability of a VLSI circuit must be considered in every phase of the chip development, from the conceptual design phase of the circuit to the final functional validation of the manufactured VLSI chip.

B. THESIS GOALS

This thesis will examine the methodology used to produce a VLSI chip from initial design to final validation of the manufactured VLSI chip, concentrating on the testing and testability issues. The development of a 16-bit correlator, NPS CORN88, will be followed throughout the phases to illustrate the process methodology.

Another goal of the thesis is the development of a program that will translate the results of a Mossim II simulation program to that suitable for use in the Tektronix DAS 9100 series tester. An introduction for the new user to the Tektronix hardware test equipment, and associated software features, will also be presented. A tutorial for the digital test facilities at NPS is presented as Appendix A.

C. THESIS ORGANIZATION

Chapter II provides the methodology for taking a design from the conceptual phase to the completion of the VLSI chip. This will include: a functional overview of NPS CORN88, its design implementation in Magic, follow on simulation with the MOSSIM II simulator, and subsequent manufacturing by MOSIS. Chapter III will concentrate on the TEKTRONIX DAS 9100 series tester and its operational capabilities. The test results of the manufactured VLSI chip, NPS CORN88, will also be examined and interpreted in this chapter. Chapter IV will present the development of the MOS2DAS translator and other support software. Chapter V summarizes the salient features of the thesis and offers recommendations for improving VLSI chip testing at the Naval Postgraduate School. Suggestions for future research are also presented. Appendix A contains a tutorial for the TEKTRONIX DAS 9100 series tester and the NPS digital test facilities. Appendix B contains the Fortran source program for the MOS2DAS translator and supporting programs. Appendix C contains the source programs necessary for generating multiple 16-bit MOSSIM II test vectors and calculating the expected results.

II. DESIGN IMPLEMENTATION

How does a VLSI chip designer advance from the design phase, through the developmental stage, to having the chip manufactured, using the facilities at NPS? What methodology is considered at each of the major phases in this process? Figure 1 on page 7 illustrates this processing flow from design to receipt of the manufactured VLSI chip. This chapter provides an overview of each of the major phases involved in the development process. Discussion of each of these phases is augmented by following the development of a 16-bit correlator, NPS CORN88. Examples are provided to highlight the methodology and salient features at each development phase. Testability issues at each phase will be addressed within that phase. The understanding of each phase of development will allow for a smooth transition between phases.

A. NPS CORN88

1. Initial Design Methodology

A correlator takes a binary string from a data source and compares that string with a reference binary string. The results of this are either an exact correlation, where each digit of the data string matches that of a reference string, or some subset of that exact correlation. Because of the many and varied applications that can be performed by a correlator, NPS CORN88 was designed to be as general as possible and produces the number of string matches as a binary coded decimal (BCD) output.

The initial iteration of this design effort [Ref. 13] was accomplished by Beck and Galinas, students at NPS. They produced a functional, stand-alone, working prototype of a correlator at the Magic architecture level. An additional design team composed of Walt Corliss, Michael Roderick, Yong Ha Ko, and David Carleton, led by Professor Chyan Yang, optimized the layout of the design, added the pad frame and associated routing, conducted additional testing, and submitted the finished design effort to MOSIS for manufacturing.

Throughout this second design iteration the concerns of testability and testing were addressed. Each individual cell (D flip flop, 2-1 multiplexer, full four bit adder, etc.) of the optimized design was functionally tested using the MOSSIM II and SPICE simulators. After these cells were assembled into the correlator, as illustrated in Figure 2 on page 9 by a Coordinate Free Lap (CFL) program, the completed design was divided into two sections for further testing, the sequential and combinational logic sections.

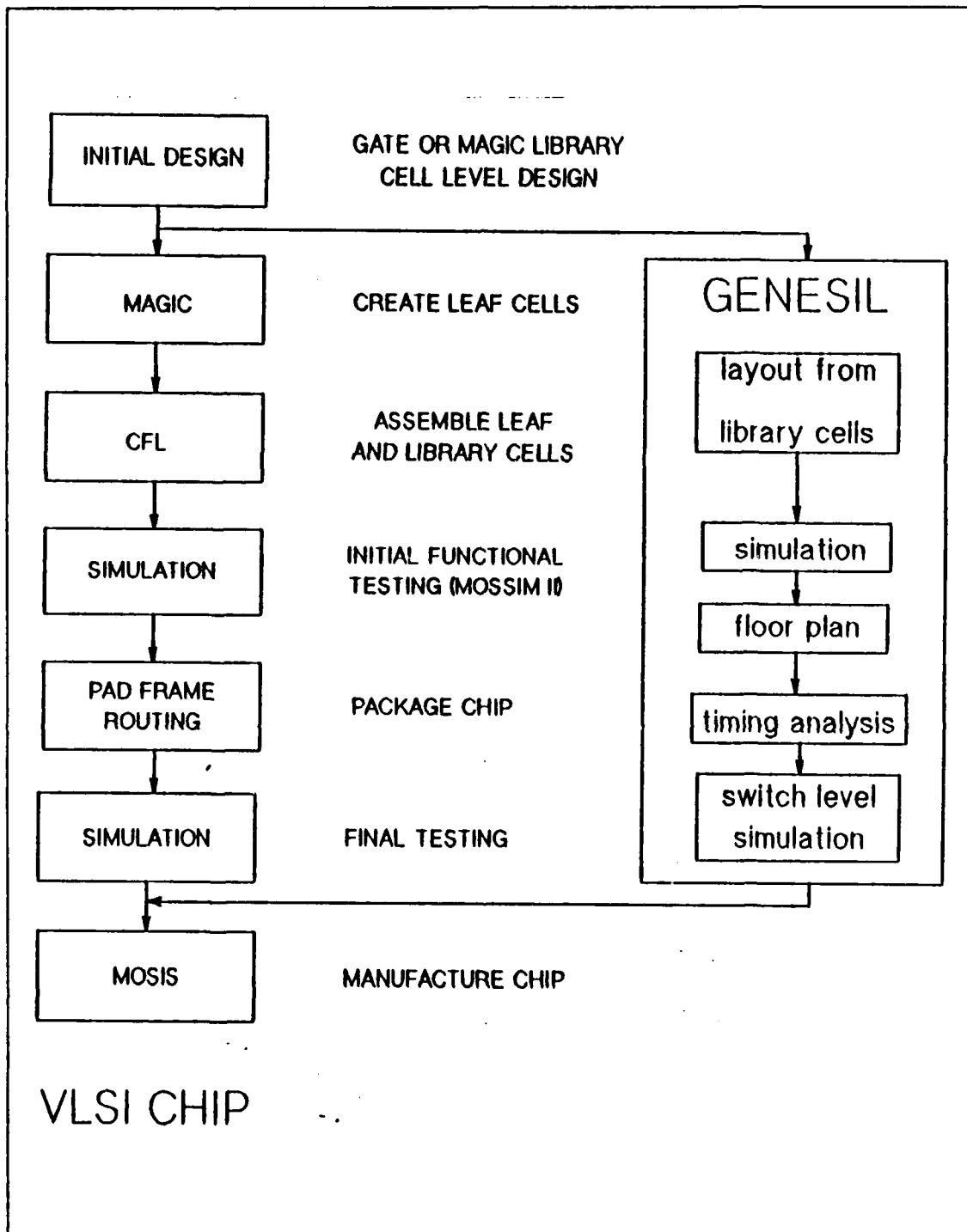


Figure 1. Design Implementation Process

After successfully testing each of these sections the entire design was tested by forcing every possible 16-bit data string combination available onto the parallel input pins. Because of the large number of 16-bit vectors that had to be generated, a C language program was written to generate 65,536 (2^{16}) different vectors. Another program was written to calculate the output for each input vector. A C-Shell program was used to run the MOSSIM II simulation with this exhaustive list of vectors and compare the correlator outputs to that of the calculated outputs. The source code for these programs are listed in Appendix C. After the successful completion of this exhaustive test, the design and simulation program were considered validated.

2. Functional Description

As discussed previously, NPS CORN88 is a 16-bit correlator that compares a data binary string with that of a reference string and outputs the BCD number of matches. As illustrated in Figure 2 on page 9, the correlator can be divided into the combinational and sequential logic sections.

The sequential logic area consists of four 16-bit register banks and various control signals. Table 1 on page 10 is a summary of the input and control pins that are contained in this section. Each of the four 16-bit register banks are identical and hold data, reference, masking and test binary string information. The register bank consists of 16 "D" flip flops controlled for either serial or parallel input by a 2-1 multiplexer and the serial parallel control (SPCON) line. In the parallel load mode the data string is loaded into the register bank that is selected by that control line. The active control line allows the clock to toggle the input into the register and inhibits the other registers. In the serial data mode the register banks can be loaded either one at a time or simultaneously. If the serial control line is selected and the associated control line is activated, the data on the input serial line to each register bank is passed. The serial string is rippled from one bit to the next bit in the register bank at the falling edge of the clock. If desired, the output control line (OUTCON) can be active every 16 clock cycles to gate the output and synchronize the serial data stream. A listing of the correlator output pins is included in Table 2 on page 10.

After the register banks are loaded, the correlation process takes place at each clock cycle. The exclusive nor (XNOR) circuitry compares each bit of the data register bank and the reference register bank and if a match is obtained a '1' is present. The mask register allows the user to select the bits to be compared by blocking the output of the XNOR circuitry. If the mask register is loaded with all '1's all compared bits from

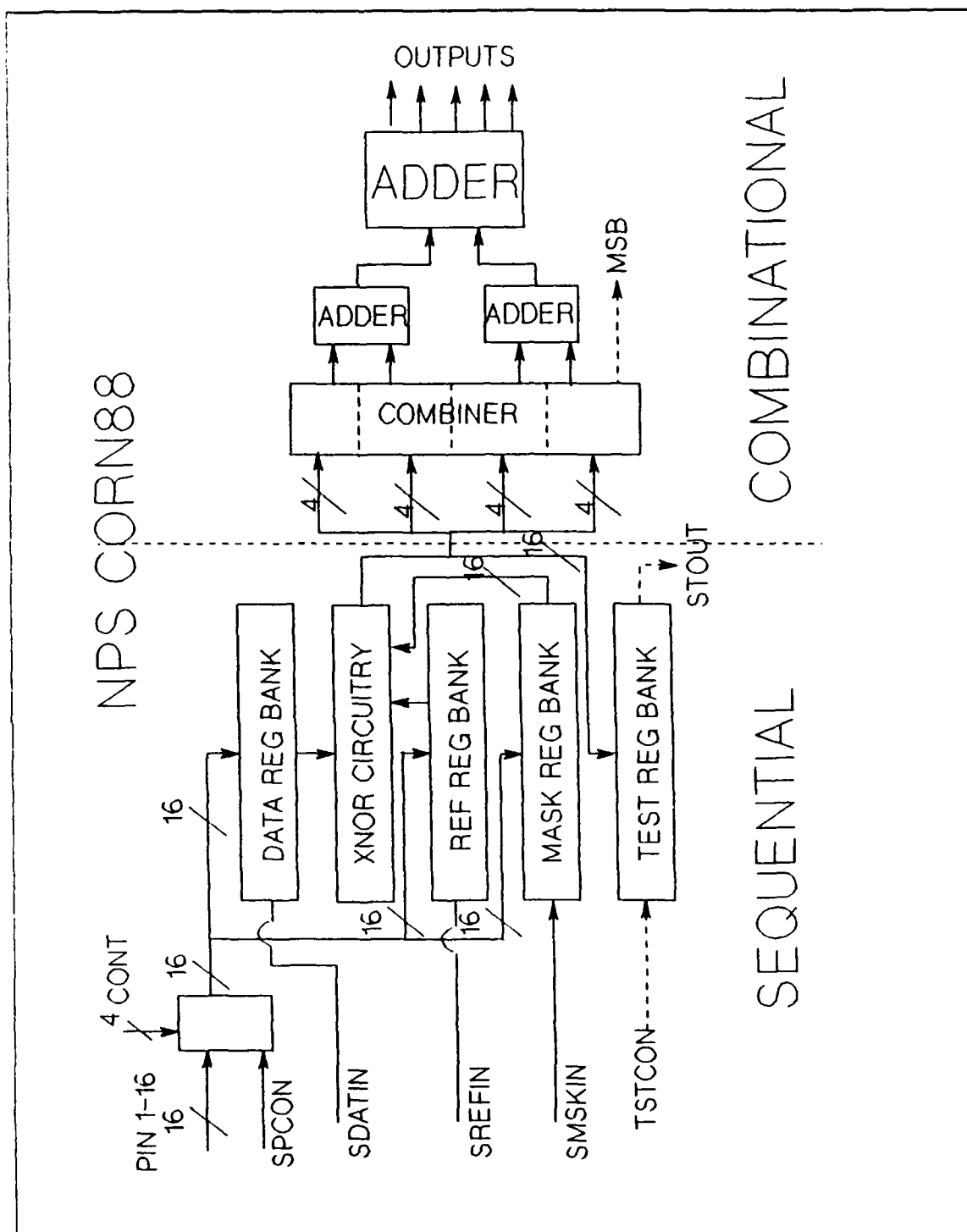


Figure 2. NPS CORN88 Block Diagram

Table 1. NPS CORN88 INPUT PINS

NAME	DESCRIPTION
In 1-16	Parallel input pins
SPCON	Serial or parallel load
DATACON	Controls DATA register bank
MASKCON	Controls the MASK register bank
REFCON	Controls the REFERENCE register bank
OUTCON	Toggles the output
CLOCK	Controls the timing of the chip
SDATIN	Serial data for DATA register bank
SMSKIN	Serial data for MASK register bank
SREFIN	Serial data for REFERENCE register bank
TSTCON	Allows the XNOR circuitry to enter the TEST register bank
PADIN	Test signal to check operation of the input pad
VCC	Power connection to the chip
GND	Ground connection to the chip

Table 2. NPS CORN88 OUTPUT PINS

Name	Description
OUT 1-5	Output coded decimal pins
SDATOUT	Serial output from DATA register bank
STSTOUT	Serial output from the TEST register bank
SREFOUT	Serial output from REFERENCE register bank
SMSKOUT	Serial output from the MASK register bank
BMSBOUT	Output of the most significant bit into the first adder

the XNOR circuitry will be passed to the input of the combiner. Table 3 on page 11, for both binary values columns, illustrates the results with the masking feature disabled.

If certain bits are not to be included in the correlation process then those bits can be masked by the mask register. With the mask register set to '1' for the bits to be compared and '0' for the remaining bits, the circuitry will pass to the combiner the number of matches from the compared bits plus the number of '0's in the remaining bits. Table 4 on page 11 column A illustrates that the masking value of '0011111111111111'

Table 3. CORRELATOR RESULTS (NO MASKING)

REGISTER	BINARY VALUES A	BINARY VALUES B
DATA REG IN	1111111111111111	0000000011111111
REF REG IN	1111111111111111	1111111111111111
MASK REG IN	1111111111111111	1111111111111111
XNOR REG OUT	1111111111111111	0000000011111111
BCD OUT DIGITAL OUT	10000 16	01000 08

will compare bits 3-16 for matches, and that number of matches plus the first two bits will give the number of '1's passed to the combiner. This method of implementing a masking scheme was chosen because, no matter how many bits to be compared, if the bits chosen match perfectly then the output will be a BCD '10000', or the decimal equivalent of 16. This allows the designer to check the output pin OUT5 to determine if a complete correlation is obtained, no matter how many bits were masked. Table 4 column B demonstrates this feature.

Table 4. CORRELATOR RESULTS (WITH MASKING)

REGISTER	BINARY VALUES A	BINARY VALUES B
DATA REG IN	0000000011111111	0000000011111111
REF REG IN	1111111111111111	1111111111111111
MASK REG IN	0011111111111111	0000000011111111
XNOR REG OUT	0000000011111111	0000000011111111
BCD OUT DIGITAL OUT	01010 8	10000 16

It is important to note that each bit of the correlator is treated the same; that is, the position of the bit is irrelevant. Serial output lines from each register bank allow for expansion of the correlator. With the addition of an external adder, two correlators can be combined and a 32-bit correlation process is possible from either serial or parallel input data. The test register bank in this section was included to improve testability and will be discussed in the next section on testability issues.

The combinational logic section contains combiner and adder circuits. The combiner is four identical logic circuits that each convert a four input signal from the XNOR circuitry to three outputs that represent a BCD number. For example, if the four input lines to the combiner were all '1's, then the three lines out of the combiner would be '100' or the decimal representation of four. The outputs of the combiners are then processed by the adder circuitry. This network is composed of three 4-bit full adders. In the first set of two full adders, each adder adds the three outputs from two of the combiners and produces a 4-bit result of the number of '1's on the input lines. The carry-in and the most significant bit to each of these two adders are set to '0'. The final adder adds the results from the previous set of adders. The final result is the five output lines which includes the carry-out pin. If the data stream matches the reference stream and the mask register bank is set to all '1's, the output from the last adder is '10000' or the decimal equivalent of 16. When the output control line (OUTCON) is active the outputs are presented at each clock cycle.

3. Partitioning for Testability

The functionality of the design was validated by the MOSSIM II simulation program, but the design offered little observability within the circuit. The initial design used 35 pins of a 40 pin pad frame package leaving five pins to access the internals of the chip for possible fault isolation. Although other VLSI chips have been manufactured for NPS by MOSIS, this was the first chip that would be tested upon return. With that in mind it was decided that adding complex circuitry to conduct insitu testing or adding redundant features was increasing the risk of producing a faulty VLSI chip. However, a test register was added to partition the combinational section from the sequential section. When the test control line (TSTCON) is raised, the input to the combiner from the XNOR circuitry is fed to the test register bank. The output of the register bank is then clocked out serially and read from the corresponding serial output. This allows for observability of the critical interface between these two sections. An input pad (PADIN) and an output pad (PADOUT) were connected without additional circuitry to validate the pad frame characteristics. The remaining pin was taken from the most significant input bit of the adder to check the output of the combiner. This partitioning of the design, while not elegant, significantly increased the observability of the manufactured chip.

B. MAGIC LAYOUT SYSTEM

MAGIC is an interactive layout editing system that was developed at the University of California at Berkeley for the design implementation of custom VLSI circuits. At NPS it allows the circuit designer, using a mouse and either the Advanced Electronic Device (AED) or one of three Integrated Solutions Incorporated (ISI) work stations, to layout circuits from the transistor device level to a full custom VLSI circuit. If transistor level design is required then knowledge of silicon semiconductor theory is a prerequisite. The goal of MAGIC "... is to increase the power and flexibility of the layout editor, so designs can be entered quickly and modified easily." [Ref. 14 ,p. 20]. Additional goals for an efficient routing capability, short turnaround time for small bug fixes and allowing the designer the ability to rearrange a cell in order to try out a different topology were also considered by the designers of MAGIC. Along with these layout goals a design-rule checking procedure to incrementally check a circuit for proper layout during the circuit creation was developed. This section will not attempt to provide the details that a designer must know to produce a design layout in MAGIC. The topographical layout procedures are described in detail in the nine tutorials provided in the MAGIC user's manual [Ref. 3]. An overview of silicon semiconductor technology, selected layout features of MAGIC, and an examination of the design-rule checking process will be discussed in this section.

1. Silicon Semiconductor Technology Overview

"A metal-oxide-silicon (MOS) structure is created by superimposing several layers of conducting, insulating and transistor forming materials." [Ref. 15,p. 5]. Figure 3 on page 14 is the palette of materials that can be combined in MAGIC to construct a VLSI circuit. The principle transistor building blocks are the ndiff, pdiff, and polysilicon materials. The ndiff material is negatively doped silicon that is rich in electrons and superimposing this material with polysilicon is interpreted by MAGIC as a nMOS transistor. A pMOS transistor is created by overlaying the polysilicon with a positively doped silicon or pdiff material. The technology that embraces both nMOS and pMOS technologies is called Complementary MOS (CMOS). MAGIC allows implementation of both nMOS and scaleable CMOS (SCMOS). SCMOS is CMOS technology that can scale the physical dimensions of the completed design prior to chip fabrication. CMOS was selected to produce NPS CORN88 because of its almost zero static power dissipation, its rise and fall transition times are of the same order, and transmission gates can be cascaded in CMOS but not in nMOS.

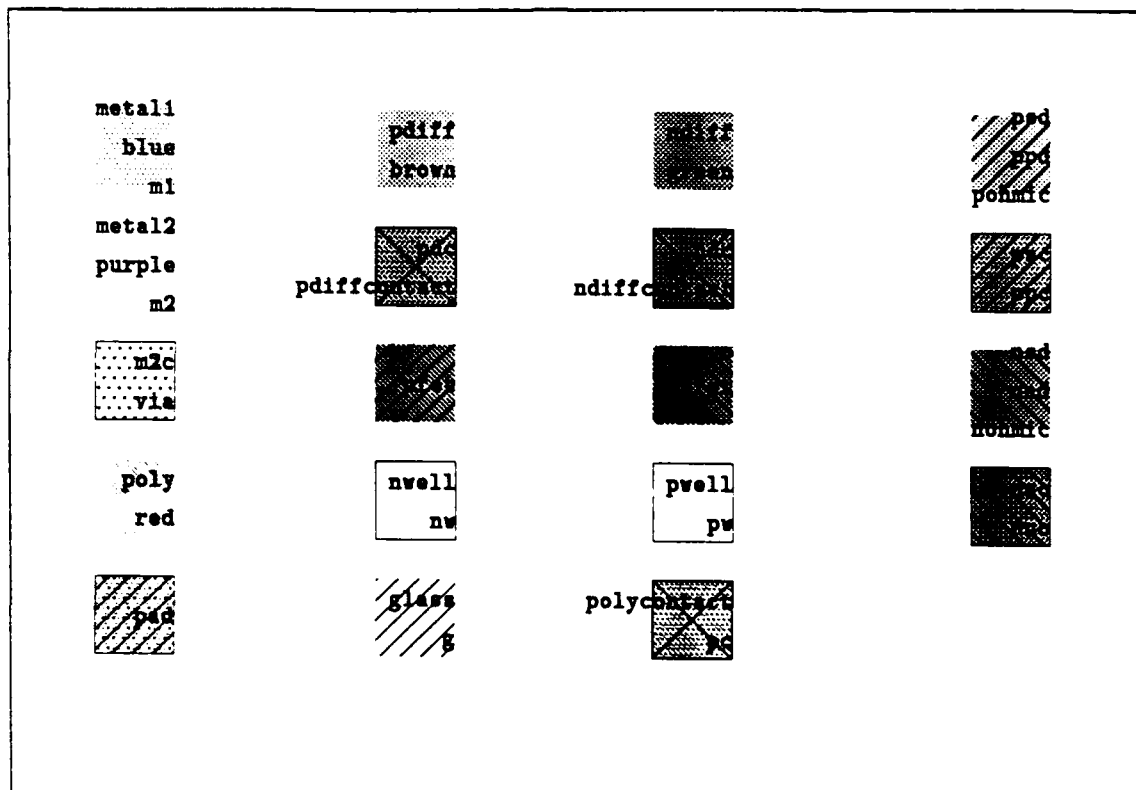


Figure 3. MAGIC Materials Palette

MAGIC layout is based on the Mead-Conway style of design which simplifies design rules and circuit structures. These simplified design rules are possible because only designs whose edges are vertical or horizontal are allowed. This design restriction sacrifices some chip density, but it is compensated for by reducing design time. An examination of a CMOS inverter, from the functional schematic to the MAGIC graphical layout represented in Figure 4 on page 15, will illustrate the basic fundamentals required in building a CMOS circuit.

The CMOS inverter is composed of an nMOS transistor and a pMOS transistor. These transistors are often referred to a N-switch and P-switch, respectively, because that is their basic function in a digital circuit application. A '0' closes a P-switch and a '1' closes the N-switch. In the CMOS inverter the P-switch is located on top and connected directly to Vdd. When a '0' is applied to the gate of the P- switch then the switch closes and Vdd is switched to the output. The '0' is also applied to the N-switch at the same time and the N-switch is turned off. When a '1' is on the input of the N- switch

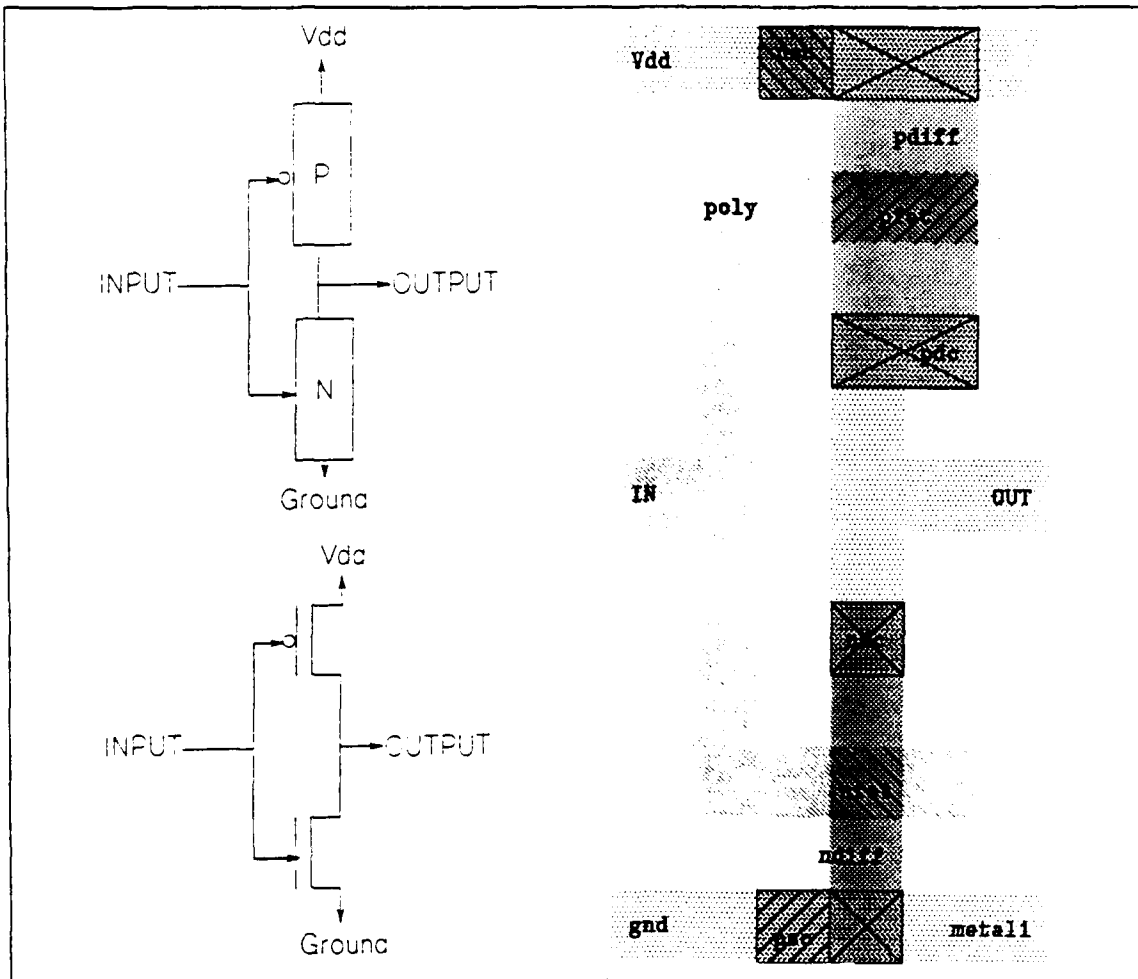


Figure 4. MAGIC Layout of a CMOS Inverter

the switch is closed and ground is passed to the output. The P-switch with a '1' on the input is turned off. In the MAGIC design of the inverter the interactive editor interprets the ndiff or pdiff overlaid on the polysilicon as a nMOS or pMOS transistor, respectively. An alternative to painting ndiff or pdiff material over polysilicon is to use the nfet or pfet paint from the palette to create the appropriate transistor. Besides the two transistors in the inverter, Figure 4 shows the metall runs of Vdd and ground. The Vdd is physically connected to the pdiff material with a contact called pdiffcontact or pdc. Conversely the ground connection to the n-diff material is called ndiffcontact or ndc. When the MAGIC design is readied for fabrication, and a CIF file is generated, the MAGIC program will cut a connection square in the mask to allow for the physical

joining of the materials. The other two contacts that are associated with the inverter located on the Vdd and ground metall runs are the substrate contacts. They are used to maintain proper substrate voltages and to prevent latch-up. The Nncontact or nnc is to provide Vdd to the n-well around the p-transistors, and Ppcontact or ppc provides ground to the p-well around the nMOS transistors.

Latch-up is a CMOS circuit parasitic feature that results in the shorting of Vdd and ground lines together because of the inclusion of both nMOS and pMOS. In the inverter example listed latch-up could occur during the transition of the input waveform. There is a remote possibility that both the pMOS and nMOS transistors are conducting at the same time channeling the Vdd directly to ground. Although this could occur for only a fraction of second it can damage the circuit over an extended period of time. Although fabrication processes have virtually eliminated latch-up problems the following design rules from Weste [Ref. 15, p. 59] further reduce the possibility of latch-up occurring.

- Every well must have a substrate contact of the appropriate type.
- Every substrate contact should be connected by metal directly to a supply pad (i.e., no diffusion or polysilicon underpasses in the supply rails).
- Place substrate contacts as close as possible to the source connection of transistors connected to the supply rails (i.e., Vss n-devices, Vdd p-devices). A very conservative rule would place one substrate contact for every supply connection.
- Place a substrate contact per 5-10 logic transistors.
- Lay out n- and p-transistors with packing of n-devices towards Vss and packing of p-devices towards Vdd. Avoid convoluted structures that intertwine n- and p-devices in checkerboard styles.

In the construction of a MAGIC cell the assembly of the materials, contacts, and routing must meet physical topographical rules. As an example of a simple size rule is that Metall width must be at least 3 units wide. The entire set of topological layout rules are checked by the design-rule checking algorithm and will be discussed in a latter section.

Additional knowledge of silicon semiconductor technology is required, when designing a VLSI circuit, to determine the timing and switching characteristics, various capacitance parasitics, power requirements and physical size of the actual devices. Based on the size of the device, for example, the length and width of the transistor, MAGIC includes the resistance and capacitances associated with that component in the extracted output. The actual mask in CIF format that is transferred to the foundry does not in-

clude these capacitances and internal resistances. These capacitances and resistances are not included because they will be realized in the manufacturing process. will be realized in the manufacturing process. These lumped capacitances and resistances allow for realistic simulation of the MAGIC design.

An example of the level of silicon semiconductor technology that is required can be shown by examining the is the previously mentioned inverter. The gain, beta, of a transistor is computed by:

$$\beta = \frac{\mu\epsilon}{T_{ox}} \left[\frac{W}{L} \right] \quad (2-1)$$

where μ is the effective surface mobility of the electrons in the channel, ϵ is the permittivity of the gate insulator, T_{ox} is the thickness of the gate insulator, W is the width of the channel, and L is the length of the channel [Ref. 15,p. 40]. The $\frac{\mu\epsilon}{T_{ox}}$ is also called the processing gain of the material. Because the processing gain of n-diff material is twice that of p-diff material, $\mu_n = 2\mu_p$, the gain of a similar size nMOS transistor is twice that of a pMOS transistor. Therefore, the width of the pdiff channel must be twice that of the ndiff channel to provide for the same theoretical gain. The rise and fall time for the inverter is also dependent on gain and will also be matched if the ratio of width of the pdiff material is twice the width of the ndiff material, $W_p = 2W_n$. Figure 4 on page 15 illustrates that ratio of widths between the pdiff and ndiff materials. The details of determining associated capacitances, power consumption, distributed resistance capacitance effects and various other silicon technology concerns are presented in Weste [Ref. 15] and should not be a concern for the beginning digital VLSI designer.

Once the basic descriptions for the P-switch and the N-switch are understood, these switches can be combined in a variety of different configurations to produce numerous sequential and combinational logic circuit designs. If the circuit design can be realized in a Karnaugh map, then these switches along with a complementary switch or pass gate can be combined to realize that circuit within MAGIC.

2. MAGIC Layout Features

a. Invoking Commands

As previously addressed, there are nine MAGIC tutorials in the users manual [Ref. 3] to acquaint the new user with the capabilities of MAGIC. Commands can be invoked in MAGIC in three different ways: mouse, macro and command lines. The

box and cursor are the principle focal points on the graphics display to manipulate items. The cursor is used to select an item and is in the form of a cross hair. The box is usually defined by the mouse's left and right buttons setting the left lower and upper right corners of the box respectively. After the box is made it can be relocated to any position by moving the cursor to the position desired and pressing the right mouse button. The center button on the mouse is used to "paint" the box the color that the cursor is touching. To "paint" is to lay materials, as defined in the palette illustrated in Figure 3 on page 14, into an area bounded by the box.

A macro is another way to control the graphics display and uses previously defined single keystroke commands. An example of a macro is typing the letter "a", which will select an item that is enclosed in the box. Macros can also be user defined to perform repetitive commands.

The third method of invoking MAGIC commands is through the command line. The command line is entered by starting the command with ":" or ";" and then proceeding with the command. An example of this type of command is ":quit" which exits the MAGIC editor.

b. Basic Magic Tools

There are three basic tools that can be utilized in the MAGIC layout editor; box, wiring and netlist. The desired tool can be selected by pressing the space bar at the ">" prompt on the editor screen. Because of its utility, the tool that the MAGIC editor defaults to is the box tool.

The box tool is by far the most useful tool within MAGIC to layout leaf cells. Using the box and cursor positions as defined by the mouse, leaf cells can be "painted" from the transistor level to the most complicated of circuits. The inverter in Figure 4 on page 15 was "painted" while in the box tool utility. This "paint" feature and the ability to label various nodes enable the designer to construct a leaf cell. Within the box tool utility a leaf cell can be copied, reoriented, moved, labeled, and modified based on which commands are invoked. The power of this utility is evidenced by the fact that all NPS CORN88 leaf cells were laid out in this box utility. This utility is also useful in the hierarchical assembly of larger circuit designs by using the ":getcell" command. This allows several cells to be viewed and manipulated on the graphics terminal in order to produce a much larger cell.

The wiring tool provides an alternative method for manipulation "paint" and is used for wiring, plowing and defining arrays. The wiring capability is just what its name implies. It provides a way to select the wire to be connected with the left mouse

button and that wire will be extended to the cursor position when the right mouse button is pressed. The wiring tool also allows cells to be copied in the "x" and "y" directions into an array of identical cells. This provides an easier way to duplicate leaf cells than was available in the box tool. The plow feature is interesting because it is used to stretch or compact cells. If it is used to compact cells it will yield the smallest possible topographical area for the cell within the design-rule limitations.

The third and final tool utility is the netlist tool. This tool makes use of MAGIC's automatic routing feature to make connections between previously defined leaf cells. A limitation of this router is that it only works between cells and not between pieces of a layout and a cell. Power and ground lines have to run to the initial cells by hand because the automatic router can not handle these cases. This automatic routing feature is useful when making a large number of interconnections between cells. If this feature is to be taken advantage of, all connections must be at the edge of the cells. A netlist is a menu driven file that describes the set of connections to make between cells. Once the netlist file has been completed the cells are automatically routed by invoking the command ":route". This tool is useful because it will route to avoid obstacles in the routing path and take the most direct path possible.

3. MAGIC External Interfaces

a. Circuit Extraction

The MAGIC extractor can be used on a single cell or a hierarchical design of many leaf cells. It converts the design layout into transistor sizes and shapes. It also recognizes nodal resistances and capacitance within the layout and includes those values in the .ext file. The extractor is relatively fast because if it is extracting a high level hierarchical design it needs only to extract those cells that have been modified and not the entire hierarchical design.

Once this design is extracted by issuing the command ".ext", the ext2sim program is used to convert this file into a .sim file. This .sim file is then used as the data file for the sim2ntk program and that .ntk file is then used as an input for MOSSIM II.

b. Caltech Intermediate Form (CIF)/ Calma Stream Format

CIF and Calma are standard layout description languages used to transfer mask-level layouts between organizations and design tools. CIF is the best known layout language in the academic community and Calma holds that distinction for the industrial community. Magic can produce either form of these output files.

CIF is invoked by issuing the ".cif" command when selecting the cell. At NPS, the CIF file is primarily used as an interface between MAGIC and the Hewlett

Packard 7586B plotter for output of hard copy of the MAGIC layout. CIF is also used to transport the MAGIC file to MOSIS for fabrication.

4. Design-Rule Checking

One of the most important features of MAGIC is the ability to incrementally check for design-rule violations during the layout phase of the design. Each of the technologies available in MAGIC has their own design-rules that reside in the technology file, and are primarily used when setting up the MAGIC file for fabrication. During the design layout MAGIC checks for design rule violations and, if any occur, it will display little white dots in the vicinity of the error. Because the editor checks for the design-rule violations incrementally when the layout is complete there is no reason to recheck the entire design. The design-rule checking feature is applicable to single cell layouts as well as larger hierarchical design layouts.

a. Single Cell Layouts

When laying out a leaf cell in MAGIC there are three basic sets of rules that the design-rule checker uses; width, spacing, and overlap [Ref. 3, pp. 255-262]. Knowledge of these rules is paramount in order to efficiently layout materials. These layout materials are listed as tile types and their abbreviations are listed in Table 5 on page 21. The design-rules for CMOS technology are summarized in Table 6 on page 21 and Table 7 on page 21. The width rule requires that any material or tile types contained in Table 5 on page 21 must meet or exceed the width specified in Table 6 on page 21 in both the "x" and "y" directions. The unit of measurement is a lambda unit and that can be converted to microns if the fabrication technology is known. NPS CORN88 used 3 micron technology for every 1.5 lambda units. For example, if a minimum width of 2 lambda units was used, that fabricated width would be 4 microns.

The spacing rules in Table 7 on page 21 are a little more complicated. The actual spacing between materials is computed by the Manhattan distance. If the points are aligned either vertically or horizontally, then the spacing is between adjacent points. For objects not horizontally or vertically aligned, the distance is the length of the longest side of the right triangle forming the diagonal between the points. Figure 5 on page 22 illustrates the Manhattan distance and how it is measured. The final column in Table 7 on page 21 indicates whether touching is "ok" or "illegal". If "touching_ok" is indicated then the materials must be either immediately adjacent or separated by the distance stated. If "touching_illegal" is indicated then that eliminates the provisions of materials to be immediately adjacent. "Touching_illegal" also checks the different layers or planes and will report those violations.

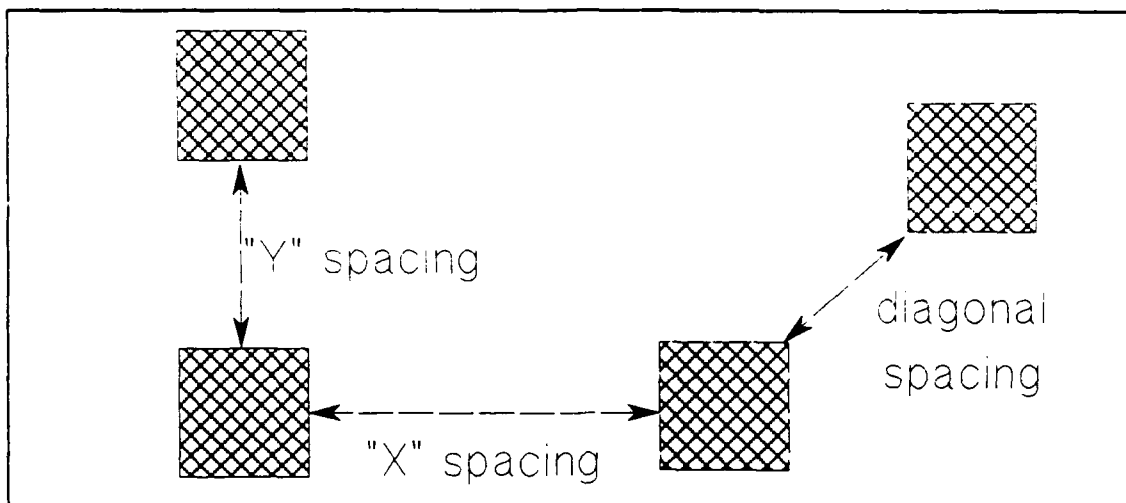


Figure 5. Manhattan Distance

Table 5. ABBREVIATIONS FOR SETS OF TILE TYPES

#define	allDif	diff,dmc,bc,efet,dfet,dcap
#define	allPoly	poly,pmc,bc,efet,dfet,dcap
#define	tran	efet,dfet
#define	contact	pmc,dmc
#define	allMetal	metal,pmc,dmc,glass
#define	allpdTypes	s,diff,poly,dmc,pmc,bc,dfet,dcap,efet

Table 6. DESIGN-RULE WIDTH REQUIREMENTS (IN LAMBDA UNITS)

width	allDif	2	<i>Diffusion width must be at least 2</i>
width	dmc	4	<i>Metal-diff contact width must be at least 4</i>
width	allPoly	2	<i>Polysilicon width must be at least 2</i>
width	pmc	4	<i>Metal-poly contact width must be at least 4</i>
width	bc	2	<i>Buried contact width must be at least 2</i>
width	eft	2	<i>Enhancement FET width must be at least 2</i>
width	dfet	2	<i>Depletion FET width must be at least 2</i>
width	dcap	2	<i>Depletion capacitor width must be at least 2</i>
width	allMetal	3	<i>Metal width must be at least 3</i>

Table 7. DESIGN-RULE WIDTH REQUIREMENTS (IN LAMBDA UNITS)

spacing	allDif	allDif	3	touching_ok
spacing	allPoly	allPoly	2	touching_ok
spacing	tran	contact	1	touching_illegal
spacing	efet	dfet,dcap	3	touching_illegal
spacing	all.Metal	all.Metal	3	touching_ok
spacing	bc	efet	3	touching_illegal

The overlap rule requires that certain kinds of overlays between materials be prohibited. Transistors can not overlap transistors of the same type. If dmc and pmc materials are touching they must completely overlap each other. These overlap rules are required for proper generation of the CIF generation and circuit extraction.

b. Hierarchical Layout Rules

There are three overall rules that govern the design-rule checker when it applies to a hierarchical layout, and each of these three rules must be satisfied independently. A hierarchical layout is built from smaller cells and NPS CORN88 is an example of a hierarchical layout. Figure 6 on page 23 shows the cells involved in the higher level design of NPS CORN88. These leaf cells were combined by a CFL program and this combination process will be addressed in a later section. Figure 6 on page 23 shows that "regcell" is the "parent" for the five "children" below. "Regcell" is duplicated in the "x" direction and stored in the parent cell "regbank". The "children", "regbank" and "comadd" are then combined into the parent "chip". This "parent" "child" relationship must be understood to examine how design-rule techniques are applied to hierarchical layouts.

The first design-rule when considering hierarchical layouts is that the "paint" in each cell must obey all design-rules by it self. The second rule involves the interface between cells. This interface could create a design-rule error because abutting connections could violate the design-rules. These errors will show up in the parent cell at the interface between the two cells. The third error is that each array must be error free by itself. If the adjoining cells interact to produce a design-rule error the violation will be presented in the parent cell.

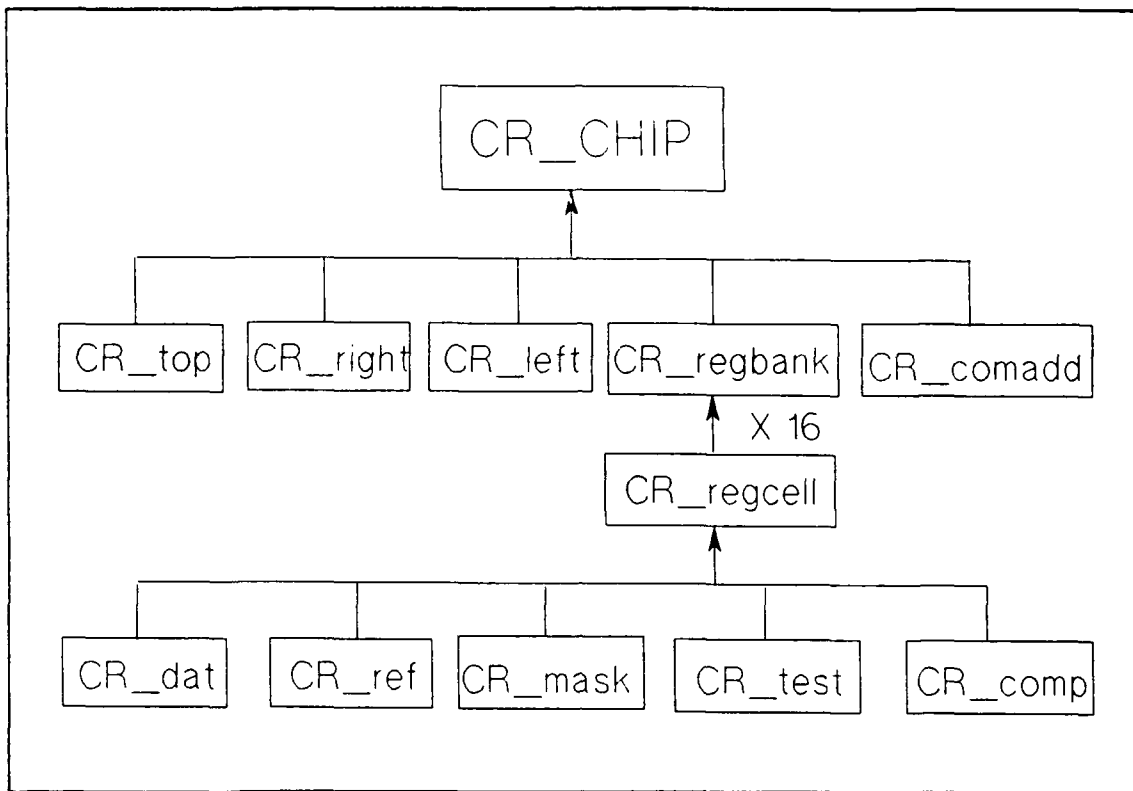


Figure 6. NPS CORN88 Hierarchical Structure

With MAGIC design-rule features the chip designer can concentrate on the chip design vice the myriad of design-rule details. If a design-rule error is discovered a simple command ":drc why" will explain the difficulty.

5. MAGIC Layout of NPS CORN88

NPS CORN88 was designed as a hierarchical layout of several small cells as illustrated in Figure 6. The cells are a combination of leaf cells from MAGIC's library of cells. For example, "CR_dat" leaf cell is a "D" flip flop and a multiplexer from the library. These two cells were assembled, modified and routed so that they would properly interface with the cells adjoining, both horizontally and vertically. This procedure was followed for the remaining subcells in the final chip assembly. The use of library cells not only saved design time but also ensured that those cells would function properly because they had already been exhaustively tested before inclusion into the library.

As each leaf cell was completed that cell was functionally verified by using the MOSSIM II and SPICE simulators. Once all the cells were laid out and functionally

tested they were assembled into the completed chip with a CFL program. This allowed for quick turnaround between the modification of a subcell and the assembly of a completed hierarchical chip layout. The ability to construct a leaf cell and test it during the incremental design process resulted in a final chip assembly that had only a few routing and alignment errors between cells.

C. SIMULATION

1. Preparation for Simulation

Once the MAGIC file has been checked for mask layout errors using the design-rule feature in MAGIC, the file can be extracted for each cell in the hierarchical design. The files that are produced in MAGIC and the paths to different simulation routines are illustrated in Figure 7 on page 25. This extracted file contains the nodes of the circuit and also includes total capacitance to ground, and a lumped resistance calculation for each node. It further defines the layout of transistors by type and size. The sim2ntk program sets up the file to be simulated, identifies the input pins and sets the strength assigned to the transistors, either CMOS or nMOS. The commands required to convert the NPS CORN88 layout design contained in the "CR_chip.mag" file, to the "CR_chip.ntk" file that is used in the MOSSIM II simulator, are included in Figure 8 on page 26.

2. Background

MOSSIM II is a logic simulator based on switch-level logic that models the MOS digital circuit as a network of nodes connected by transistor switches. Because of this logic level modeling, MOSSIM II can handle large VLSI designs with long input sequence vectors. The MOSSIM II program is written in Mainsail programming language and using the network description language (NDL) the user can fully describe a network to the simulator. If a design can be fully described in this manner, it does not have to be laid out in MAGIC to be simulated. This feature was not used in the development of NPS CORN88 and will not be discussed.

The MOSSIM II user's manual [Ref. 8] provides all the details required to use MOSSIM II. The actual commands that are involved in controlling the MOSSIM II program will be detailed in Chapter 4 along with the development of the MOS2DAS translator. An overview of MOSSIM II functional capabilities will be discussed in this section along with the applications that applied to the testing of NPS CORN88.

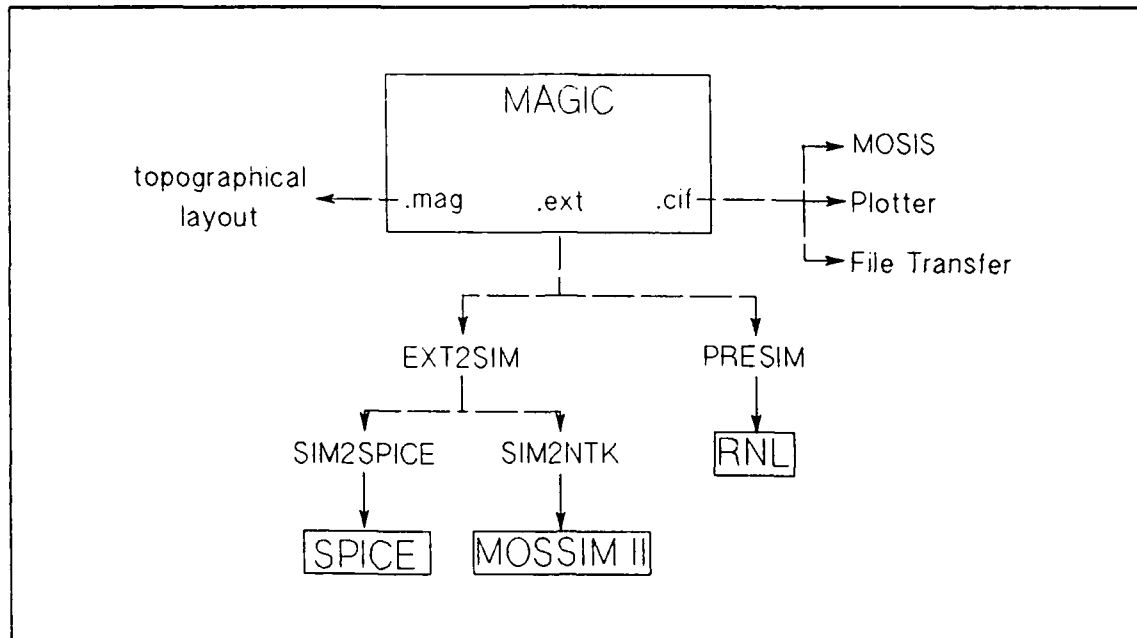


Figure 7. MAGIC to Simulation File Conversion Process

3. Overview of Capabilities

MOSSIM II capabilities can be examined by separating the source program into two different sections, initialization and drive. Figure 9 on page 27 is the MOSSIM II program that was used to simulate NPS CORN88 and will be used to illustrate the basic structure of a MOSSIM II program. In the initialize area the file to be simulated is read, and the clocking scheme, vector and watch assignments are defined. In Figure 9 on page 27 the file read is "CR_chip.ntk" and the clock is three phases, '010', with a node name of "clk". The vectors defined are "ins" for the 16 parallel input nodes, "control" for the control lines, "select" to select a serial data input, "misc" to set up the serial parallel and output control lines, and finally "outs" to define the five output pins. The watch command identifies vectors and nodes of interest that need to be observed. The vectors "outs" and "ins" and the node "clk" are "watched" in this program. Once the initialize section is complete, the simulator is ready to start handling binary assignments to the inputs.

The drive section of a MOSSIM II simulation program sets or forces the binary values to the desired inputs and also sets the number of cycles for the simulator to run. A cycle is the completion of the one clock cycle. The output values for the watch defined nodes are observed after each cycle or phase as specified. In this case the watch

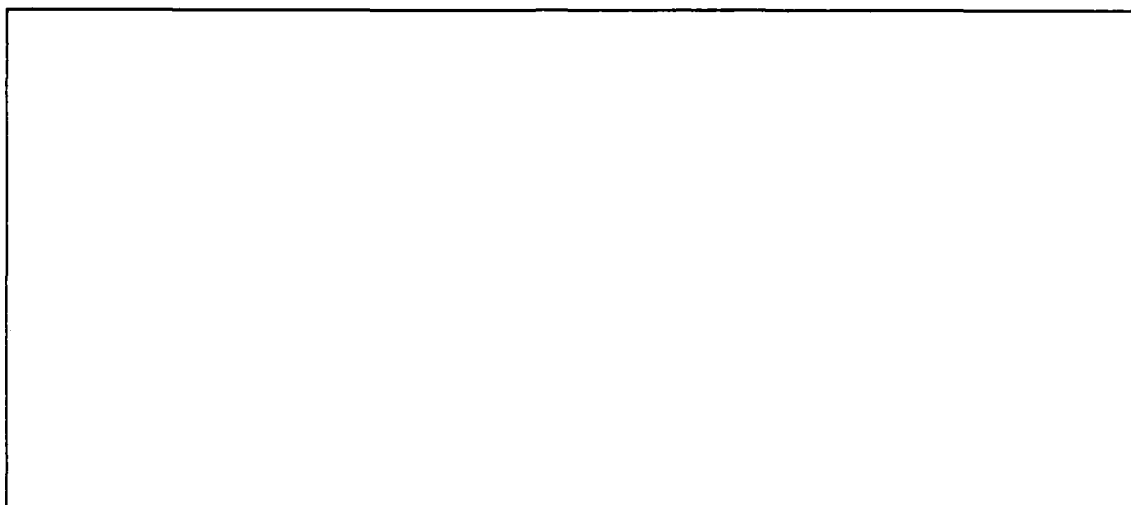


Figure 8. MAGIC ".mag" File to ".ntk" file Commands

command " #" indicates that the outputs will be observed after every clock phase, or three phases per cycle. The program in Figure 9 on page 27 forces the parallel inputs to the desired values and the values are then loaded into the respective register at the trailing edge of the clock pulse. To prevent erroneous data at the output during this register set up process, the output lines are disabled. Once all the registers are loaded, the outputs are enabled and the vector "outs" contains the BCD representation of the number of matches. The remaining program ripples an error bit into each bit of the data register to verify proper operation of the correlator. The output line reports the cycle number and the nodes of interest. An "X" state is an undefined state and will occur at a watched node if that part of the network has not been initialized or the node voltage lies between the logic thresholds.

Additional features of the MOSSIM II simulator can be obtained by setting "switches" to turn on or off. Four of these switch positions are helpful in run time testing of the circuits. These run time switches are summarized below [Ref. 8,p. 18]:

- **Ternary**--provides a rigorous test to discover possible timing errors. If a design simulates successfully in the ternary mode the circuit will function correctly for those input sequences regardless of the actual switching and communications delays of the circuit. This switch must have a two phase, non-overlapping clocking scheme as the design clock, and will fail for clock generators and edge triggered flip flops. This switch can also determine race conditions within a circuit.
- **Restore**--checks for and reports unrestored CMOS logic levels.
- **Explain**--the simulator makes an attempt to diagnose the reason for the "x" state.

```

>read CR_chip.ntk
#1
1386 nodes, 2939 transistors, 0 blocks
>
>initialize
>clock clk:010
>vector ins in1 in2 in3 in4 in5 in6 in7 in8 in9 in10
        in11 in12 in13 in14 in15 in16
>vector control dcon rcon xcon tstcon
>vector select sref sdat sxnor
>vector misc spcon outcon
>vector outs out5 out4 out3 out2 out1
>watch /* outs
>watch /* ins clk
>
>comment load reference data
>force ins:1111111100000000
>force control:0100
>force select:000
>force misc:10
>cycle 1
  1.1 | outs:00000 ins:1111111100000000 clk:0
  1.2 | outs:00000 ins:1111111100000000 clk:1
  1.3 | outs:00000 ins:1111111100000000 clk:0
>
>comment load bit select data
>force ins:1111111111111111
>force control:0010
>cycle 1
  2.1 | outs:00000 ins:1111111111111111 clk:0
  2.2 | outs:00000 ins:1111111111111111 clk:1
  2.3 | outs:00000 ins:1111111111111111 clk:0
>
>comment load signal data
>force ins:1111111111111111
>force control:1000
>force misc:11
>cycle 1
  3.1 | outs:XXXXX ins:1111111111111111 clk:0
  3.2 | outs:XXXXX ins:1111111111111111 clk:1
  3.3 | outs:01000 ins:1111111111111111 clk:0
>
>comment changes mask register
>force control:0010
>force ins:0000000011111111
>cycle 1
  4.1 | outs:01000 ins:0000000011111111 clk:0
  4.2 | outs:01000 ins:0000000011111111 clk:1
  4.3 | outs:01000 ins:0000000011111111 clk:0
>

```

Figure 9. NPS CORN88 MOSSIM II Source Program

- **Statistics**--will print run time statistics including number of steps, CPU time required to run and other timing details.

4. NPS CORN88 Test Vectors

The selection of test vectors to properly stimulate the circuit design is an important aspect in the simulation procedure. As mentioned previously a C program was written to simulate 65,536 (2^{16}) possible test vectors. An additional program verified that the correct values for each test vector was obtained. These programs took more than 23 hours to simulate on the VAX 11/785. While this exhaustive method provided the maximum test coverage, a subset of these vectors was found to provide acceptable coverage with far fewer test vectors.

This subset of vectors was heuristically determined and included only 63 vectors. The first 16 vectors rippled an error through the data bits and verified a BCD output of 15 was observed at the output. The second set of 15 vectors insured that the output would produce BCD values from 0-16. Only 15 vectors are needed because the BCD value for 15 was determined in the first case. The next two vector sets involved changing the reference register and the mask register. Setting the data register with all ones an error was rippled through the reference register bits and the output observed. The mask register was similarly tested with 16 additional vectors. These 63 test vectors functionally validated NPS CORN88 and provided a reasonable set of test vectors for MOSSIM II, and the follow on testing of the fabricated chip.

D. CFL

"Coordinate Free Lap (CFL) is a library of subroutines written in C intended to facilitate the construction of VLSI circuit layouts." [Ref. 16]. Once the required leaf cells have been laid out in Magic, then the CFL algorithms provide powerful tools to duplicate, align and combine these cells into the assembled chip. The CFL routines were used twice in the development of NPS CORN88, once to assemble the body of the chip, Figure 10 on page 29, and in the final assembly process to attach the pad frame and associated routing. The description of CFL procedures, format and symbology are delineated in the CFL reference manual, [Ref. 16], but discussion of the salient features of Figure 10 on page 29 will illustrate the strengths of this program.

1. Hierarchical Assembly of Magic Leaf Cells

As previously mentioned Magic is used to generate the leaf cells that are assembled with CFL to produce a composite chip. The leaf cells used by Figure 10 on page 29 include the four register cells and an XNOR cell. Each of the register cells

```

#include <stdio.h>
#include <time.h>
#include "cfl.h"

main()
{
    SYMBOL *chip;

    cflsetc("format","magic");
    cflstart("scmos");
    cflsetv("grain",1);

    chip = ll(gs("CR_comp"),gs("CR_dat"));           line 01
    chip = ll(gs("CR_ref"),chip);                   line 02
    chip = ll(gs("CR_mask"),chip);                  line 03
    chip = ll(gs("CR_test"),chip);                  line 04
    ps("CR_regcell",chip);                          line 05
    chip = nx(gs("CR_regcell"),16);                 line 06
    ps("CR_regbank",chip);                          line 07
    chip = ll(gs("CR_comadd"),gs("CR_regbank"));    line 08
    chip = ttdy(gs("CR_left"),chip,27);             line 09
    chip = rr(chip,gs("CR_top"));                   line 10
    chip = ttdy(chip,gs("CR_right"), -7);           line 11

    ps("CR_chip", chip);
    cflstop();
}

```

Figure 10. CFL Composite Chip Assembly

("CR_dat", "CR_ref", "CR_mask", "CR_test") includes a "D" flip flop, 2-1 multiplexer, and control lines. "CR_comp" contains the XNOR circuitry. After the initial setup commands to link the program to the CFL library the first construction command lies on line 01. Line numbers have been added to the program in the comments column to refer to them in this discussion and are not normally part of the program. Line 01 uses the "gs" command to get the library symbols "CR_comp" and "CR_dat" and the "ll" (lower case LL) command aligns them left border to left border with the "CR_dat" cell on the top. This new cell is stored as "chip". Figure 11 on page 31 illustrates the CFL construction of the "CR_regcell" after lines 01-05 are completed and the "ps" command is used to put "CR_regcell" into "chip". Line 06 duplicates the "CR_regcell" 16 times in

the "x" direction and line 07 stores that symbol as "CR_regbank". The remaining lines align the remainder of the leaf cells to the basic chip. The combiner and adder leaf cell, "CR_comadd", was built as a large cell because the interconnection between the components were difficult to realize in CFL. It was easier to build the cells in Magic and then duplicate and connect them while observing the results on the Magic screen. The designer must determine if the cell can be replicated more easily in Magic or by using the CFL program.

2. CFL Advantages/Disadvantages

The obvious advantage for using CFL is that the individual leaf cell can be modified in Magic and the CFL program can assemble the chip with that modified cell. If a cell was modified in the Magic drawing (for example, one of the register cells), each cell of the Magic drawing would have to be modified, or 16 modifications. This could create errors in the design and also is much more time consuming than using CFL. Use of CFL reduces the development time considerably and new modifications of the cell can occur expediently. The disadvantage of using CFL is that the cells must be constructed block-like for ease in alignment during assembling routines. Each interface that is associated with that leaf cell must be designed with the adjoining leaf cells in mind. "CR_regcell" in Figure 12 on page 32 is a good example of well constructed leaf cells for use in the CFL program. The cell is "finished" to each border. Notice how the outputs of each cell must line up with the adjoining cells inputs. The other disadvantage of using CFL is that of labeling external connections for the chip. If a leaf cell is built with labels, those labels will be duplicated, when the leaf cell is duplicated and any attempt to run a simulation program using those labels as inputs or outputs would not succeed because of label duplication. A frame must be built that contains connections to the composite chip that includes labels. In Figure 10 on page 29, "CR_left", "CR_right", and "CR_top" provide those connections and associated labels.

In summary, CFL is a powerful assembly tool for leaf cells but not without a price. The extra effort expended in designing the leaf cell to align properly with adjoining cells, and providing leaf cells that contain labels and associated connections, could be considerable and not worth the effort. NPS CORN88 was a combination of both assembly technologies. The combiner adder circuitry was constructed in Magic as a large leaf cell because the individual modules were easier to connect on the Magic screen. The individual register and XNOR leaf cells were designed in Magic but assembled into a 16-bit register bank using CFL.

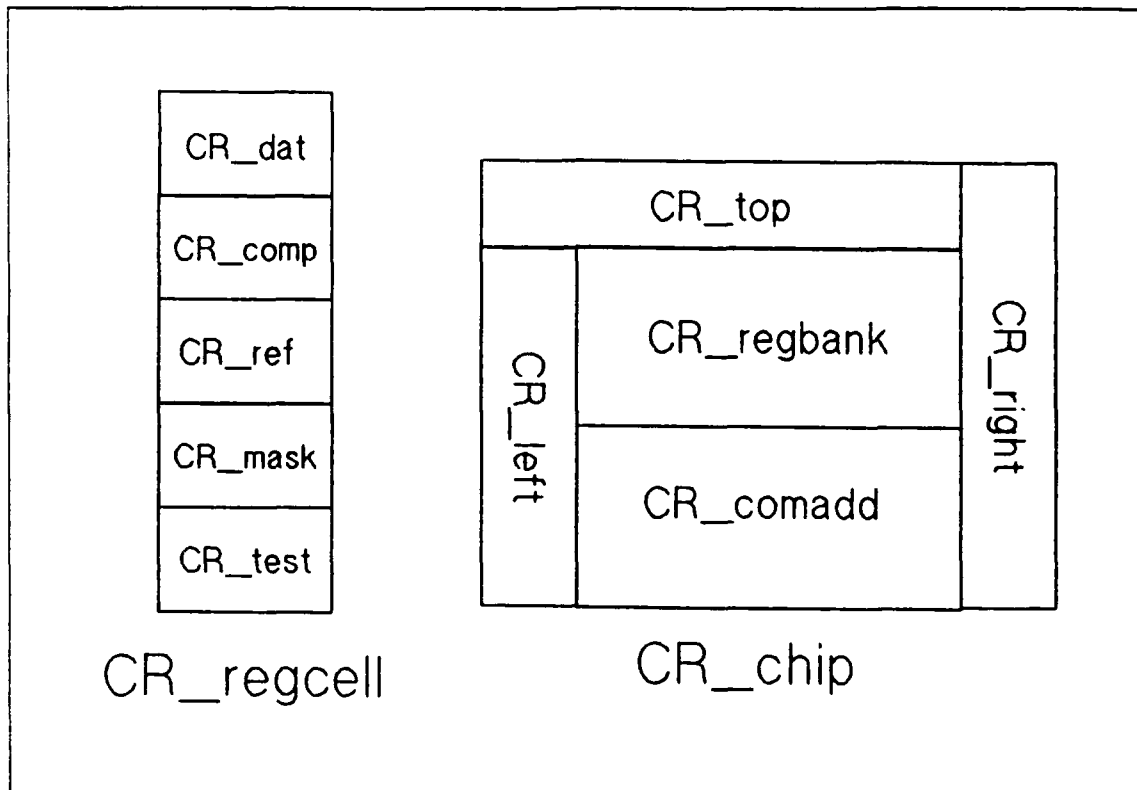


Figure 11. NPS CORN88 CFL Block Structure

E. MOSIS

1. Background

MOSIS, which is an acronym for MOS Implementation System, provides fabrications services for the production of bonded and packaged integrated circuits. Originally it provided services to government and educational users but recently it has expanded to the industrial sector. MOSIS, as described in the MOSIS user's manual [Ref. 11], is able to combine several layout designs from different users of the same requested technology onto a multi-project single silicon wafer and thus dramatically reduce the fabrication cost. Tomovich states, "Instead of paying for the cost of maskmaking, fabrication, and packaging for a complete run (currently up to between \$50 and \$60,000), a designer can receive packaged parts for as little as a few hundred dollars." [Ref. 17]. Before MOSIS can fabricate the chip, several steps are required to prepare the final design for fabrication.

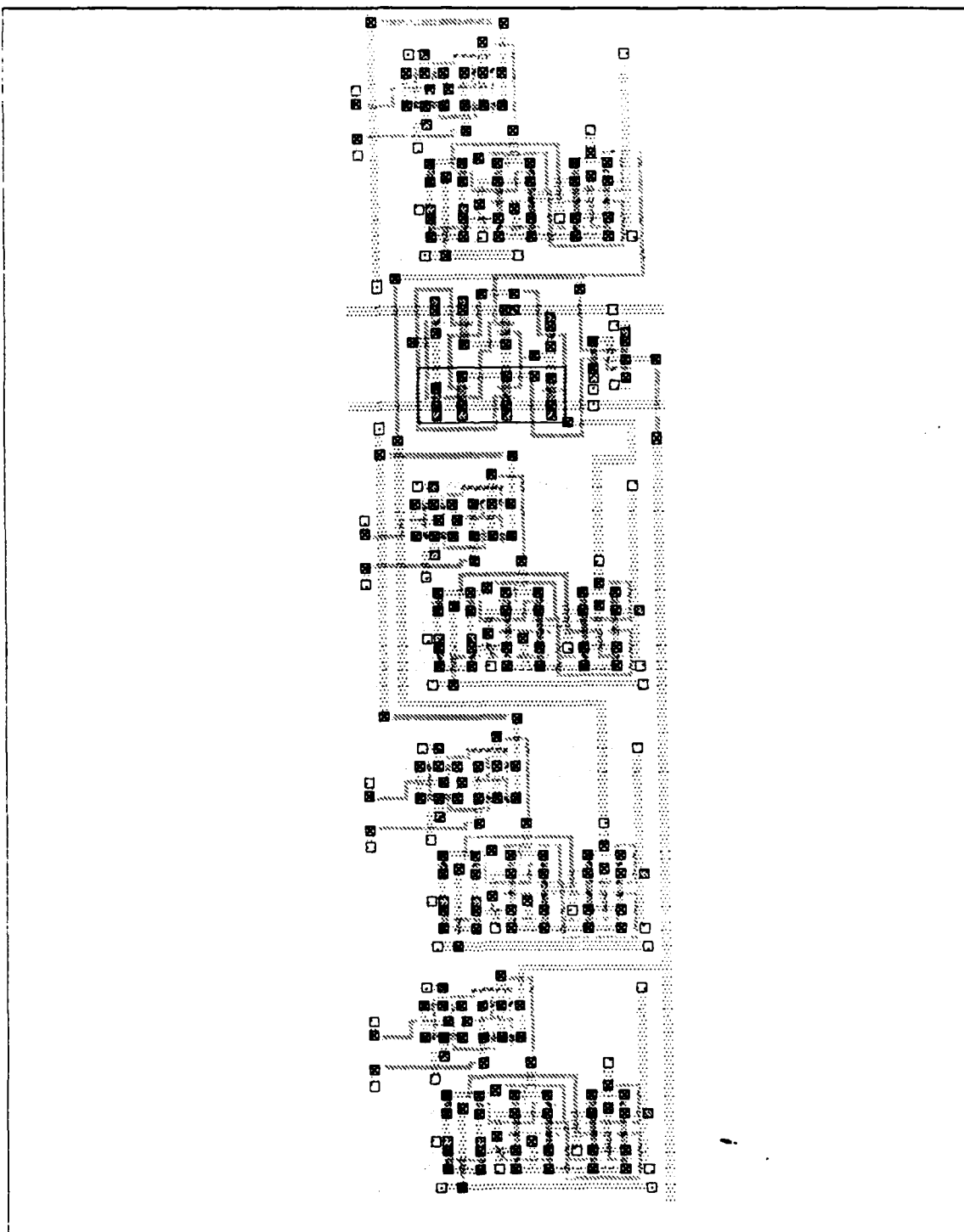


Figure 12. CFL CR_regcell

2. Preparations Prior to Fabrication

Once the circuit design has been validated, the designer must choose a pad frame to encase the design. There are several standard pad frames recommended by MOSIS and available from their library of cells. The frames come in 28, 40, 64, 84, 108, 132 pin packages and they facilitate the entire packaging and bonding process. It is recommended that one of these standard frames be used along with the SPADS program to generate the necessary pads and associated frame. Carleton describes the procedures in producing the pads for NPS CORN88 [Ref. 18]. In all but one instance (the frame with 84 pins), all standard frames have a substrate connection at pin #1. It is important to rotate the design to enable this pin to be used as Vdd. The associated routing from the basic design to the pads is done by a CFL program, whose methodology was described earlier. When the entire chip is assembled in a MAGIC file and has been properly validated, the file then is converted to the Caltech Intermediate Form, CIF, format. This is done by loading the entire design on the graphics terminal and typing the command ":cif" at the prompt. This CIF file is also the form that is needed to print the file on the NPS Hewlett Packard 7586B plotter.

Now that the CIF file is ready, an account has to be established with MOSIS to pay for the manufacturing of the chip. "Universities that are teaching classes in VLSI design may apply for government-sponsored use of MOSIS using the MOSIS application form for Education." [Ref. 11,p. 12]. NPS CORN88 was funded through this method with an account set up by National Science Foundation (NSF).

3. MOSIS Quality Control

Once an account has been established, the designer can send the CIF file to MOSIS via the Internet, formerly known as the ARPANET. After MOSIS receives the CIF file it will check that file for syntax errors and thereby validate the syntax. A program, CIF-CHECKSUM, can be run on the user system and sent to MOSIS to verify that the file as transmitted was complete. MOSIS does not validate the design; it only ensures that the CIF file is in the proper syntax and no errors were made in the transmission of that file.

MOSIS also monitors the quality of the fabrication foundry, the mask vendors and the packaging facility. The mask vendors receive the multi-project tape for E-beam tooling and the resulting mask must meet tight defect specifications. MOSIS installs several designs on each of the multi-project wafers to act as process monitors. The yield and parametric specifications for the process monitors are checked by both the vendor and MOSIS before the wafer is accepted. Once the wafer is accepted it is sent to the

packager for assembly into the dual in-line package, DIP. "Packagers are qualified on the basis of bond pull tests and must follow strict static electricity control procedures in handling user's projects." [Ref. 11,p. 2]. When the packaging specifications are met a set of bonded and packaged integrated circuits are returned to the user. Mosis produced twelve NPS CORN88 custom VLSI chips in six weeks at the modest cost of \$1400.00.

III. NPS VLSI DIGITAL TEST FACILITIES

A. BACKGROUND

As digital technology has matured and become more complex, the capability to test these circuits have become more sophisticated. The advent of CAD tools to design circuit layouts has lead to more full custom VLSI design implementations. Each of these custom VLSI designs requires a unique test program to functionally validate both the design level and fabricated chip. Hnatek states that, "Testing is rapidly becoming the largest portion of recurring costs in the VLSI device manufacturing process." [Ref. 19,p. 22]. The NPS CORN88 design was previously validated by the MOSSIM II simulator, fabricated by MOSIS, and will be used as an example throughout this chapter.

This chapter will examine the digital test facilities at NPS and is separated into three sections. The first of these sections will examine the Digital Analysis System (DAS) model 9100 and its capabilities and limitations. The second section will detail the Device Verification Software (DVS) model 50 functions and its interface with the DAS 9100. The final section will discuss the set-up and testing of NPS CORN88.

B. DIGITAL ANALYSIS SYSTEM (DAS) 9100

1. Overview

The DAS 9100 is a stand-alone, modular assembly of various configurations that can perform selected pattern generation and data acquisition functions. This modularity is engineered into the DAS 9100 mainframe by allowing the user the ability to install different data acquisition and pattern generation modules. The DAS 9100 Series operator manual [Ref. 12] details the numerous different module options available as well as providing an in-depth description of the DAS 9100 functions and capabilities. Table 8 on page 36 reflect the modules available for the DAS 9100 and summarizes their capabilities. Up to six of these modules can be installed in the DAS 9100 mainframe.

The DAS 9100 can be used as a logic analyzer, as a pattern generator or as a combination of both. One of the advantages of the DAS 9100 is that its operations are controlled by interactive menus. The data acquisition and pattern generation modes of operation, and their respective menus, will be discussed in this section and is only an overview of the information contained the DAS 9100 series operator manual [Ref. 12]. The external interfaces from the DAS 9100 will also be addressed in this section. The

Table 8. DIGITAL ANALYSIS SYSTEM (DAS) CAPABILITIES

MODULE	TYPE	# CHNS	SPEED	MAX INST
91S16	Pattern Generator	16	50MHz	1 1
91S32	Pattern Generator	32	25MHz	6 1
91A08	Data Acquisition	8	100MHz	4 0
91A32	Data Acquisition	32	25MHz	3 1
GP1B	Interface to DVS50			1 1
RS232	Connects 2 DAS units			1/1
Hardcopy Interface	Printer			1 1
Tape Drive	Data Storage			1 0

actual connections to the device under test (DUT) and the test results will be included in a latter section.

2. Data Acquisition

a. Modules

The maximum number of parallel data acquisition channels that can be processed by the DAS 9100 is 104 at one time. This configuration is obtained by installing three 91A32 modules and one 91A08 module. Some discussion of each of these modules is necessary to examine the capabilities of the DAS 9100.

The 91A32 can acquire 32 channels at an internal or external clock interval ranging from 40ns to 5ms or a maximum clock rate of 25Mz. Each 91A32 module installed can be run at a different clock rate and contains two clock qualifiers. These clock qualifiers are used by the DAS 9100 to provide a condition on storing the data acquired into memory. If both clock qualifier lines are used, and if those lines meet preset conditions, then the data acquired is stored in memory. Each 91A32 module controls four data acquisition pods that each contain eight probes to hook to the DUT.

The DAS 9100 can support up to four 91A08 modules and each can acquire and store eight channels. This module, with a special clock probe, can acquire data at a rate of 100MHz. This module can also use a internal or external clocking scheme and each module has a clock qualifier line. The 91A08 module can also acquire and store

glitches on all acquisition channels. A glitch is defined as two or more transitions occurring within any given clock cycle.

The two data acquisition modules, 91A32 and 91A08, are compatible and can be used at the same time within the mainframe. The first of these combinations is the **AND** mode and operates both modules simultaneously at the clock rate set for the 91A32. The **ARMS** mode allows different clock rates for the modules, with the 91A32 enabling the 91A08 trigger to produce a display similar to an oscilloscope sweep.

b. Menus

During the logic analysis function of the DAS 9100, the data acquisition modules acquire and store digital patterns generated by the DUT. Five interactive menus on the DAS 9100 are used to set up the clocking scheme, specify channel assignments, determine trigger specifications and display the data. Each menu is detailed in its own section of the operator's manual [Ref. 12] but an overview is provided to illustrate the logic analysis function of the DAS 9100.

The **Channel Specification** menu controls the display format for the incoming data to be used by the **Trigger Specification**, **State Table**, and **Define Mnemonics** menus. The **Channel Specification** menu also specifies the input voltage thresholds to be used by the probes and the which defaults to TTL voltage levels. Each data acquisition probe has a threshold switch physically located on the back of the probe that can switch to "NORM" or "AUX". The "NORM" position lets the **Channel Specification** menu choose between the TTL voltage threshold levels or a "VAR" threshold that can be set between -2.5V to +5.0V. If the "AUX" switch is chosen on the probe then the threshold voltage can be set to MOS voltage ranges, which can be between -10.0V to +20.0V. The "NORM" or the "AUX" position switch settings can not be mixed in each pod.

The **Trigger Specification** menu is used to set up data acquisition and storage conditions. The menu defines the data acquisition modules to be used, determines the clock rate, sets up clock qualifiers and word recognition patterns. Each of the modules has its own sub-menu to control the trigger specification. The clock qualifiers set parameters on when to store data acquisition. The word recognition feature allows the DAS 9100 to set a trigger for each occurrence of the word in the data acquisition process. This makes it relatively easy to find a high interest pattern or to use that trigger to start another set of pattern sequences. Usually, both the clock qualifier and word recognition features are set to the "don't care", "X", symbol to allow storage of all data acquired and not set the trigger on the word recognition feature.

The sub-menu for the 91A08 does everything that the 91A32 does and it can set the trigger on glitches. The 91A08 always acquires and stores the glitches, but if the trigger is set "on" for glitches it is easier to find where the glitches occurred. This menu also sets up whether the **ARMS** or **AND** clocking scheme is to be employed.

The **State Table** menu provides access to the data acquired and reference memories. This menu can show either the data acquired from the sub-menu acquisition, the reference memory from the sub-menu reference, or a comparison of both reference and acquired data. This menu is useful when comparing two files but the information is in hex format and not waveform display so it can be tedious to compare.

The **Timing Diagram** menu displays the most recently acquired data in a standard digital waveform format for both the 91A32 and the 91A08 modules. Up to 16 channels can be displayed and this menu can magnify the display, step through the entire memory of acquired data a "window" at a time, place a cursor at a run time of the pattern and read out the digital values of all channels acquired. The waveforms correspond to the pod that acquired the data and names can be added to the waveform display to make it more intelligible.

The final menu associated with data acquisition is **Mnemonics**. This menu can generate the set up for the **state table** display. Several different tables can be defined, and then the data acquired can be presented in format of that table. This is particularly useful if the user wants to observe several different formats of the acquired data.

The data acquisition features of the DAS 9100 are impressive but it can be tedious to enter the specifications via the DAS 9100 keyboard. The menus make it somewhat user friendly, but a study of the user's manual is required to use data acquisition on even the simplest circuit. The DAS 9100 at NPS lacks a tape recorder to save all the set-up data and reference waveforms. These must be initialized every time the DAS 9100 is powered up. This is not only tedious and inconvenient, but it is an excellent opportunity to make input errors. Nevertheless, the data acquisition function of the DAS 9100 does provide the user with a valuable tool to analyze the fabricated chip.

3. DAS 9100 Pattern Generation.

The DAS 9100 generates clock and data signals to the DUT in the pattern generation mode. Two pattern generator modules are available, 91S16 and 91S32. The clocking rate for either module is limited to 25MHz and can either be internally or externally generated. If all six modules are configured to 91S32 pattern generation modules the DAS 9100 can provide 192 channels of patterns. This would leave no slots available for data acquisition, so the practical limit for pattern generator modules, if only

one DAS 9100 is used, is less. Patterns can be generated in either of two ways. The 91S16 is an algorithmic pattern generator that uses a nine program instruction set to provide branching, looping, and other algorithmic patterns for 16 channels. Only one 91S16 can be installed in the DAS 9100, but it can be used as a controller for the installed 91S32 modules.

Each 91S32 module generates 32 channels of RAM-based patterns. This feature requires the user to input each channel sequence into RAM memory. If the 91S16 and 91S32 modules are installed together, the 91S16 controls the master cycle clock. There are two operating modes when the two pattern generator modules are to be operated simultaneously, sequential and follow 91S16.

In the sequential mode each module operates separately but the 91S16 supplies the clock signal. When the 91S16 algorithm reaches the end of its program it will recycle until the pattern sequence generated by the 91S32 is completed. In the follow 91S16 mode the 91S16 has active control of the pattern generated by the 91S32 and can algorithmically control the pattern sequence.

The difficulty in using the DAS 9100 for pattern generation is the same difficulty that was mentioned for the data acquisition feature, the tedious manner that the input must be specified. While 16 channels can be generated by an algorithm controlled by the 91S16 module, the 91S32 pattern sequences require each channel to be individually defined by use of the keyboard. This could be a very long process if a long pattern sequence was required for many channels. An interface with a tape recorder, or another memory storage device, that could save and load these long patterns once the power was shut down would be invaluable.

An important feature of the DAS 9100 is the ability to run a pattern sequence test pattern and compare until the next sequence either compares or do not compare. As an example, the initial results of the NPS CORN88 test sequence were stored in the compare file. The test was rerun and compared with the original test until an error was found. This enabled many test runs to be compared to each other to run a much longer test.

4. External Interfaces

There are four external interfaces defined in the DAS 9100: General Purpose Interface Board (GPIB), RS-232, Tape drive, and Printer. The GPIB interface allows parallel data transmission between a host computer and the DAS 9100, which acts as the talker and listener. The NPS test configuration is set-up so that a personal computer

(PC) running the DVS50 software program can provide the inputs to the DAS 9100 and display the data acquired from the DAS 9100 via the GPIB interface.

"The RS-232 interface allows two DAS 9100 systems to be linked together for a master slave transmission." [Ref. 12, pp. 1-3]. This allows the master DAS 9100 to control the acquisition of data and pattern generation of another DAS 9100 unit. A printer interface is provided to produce hard copy results for latter analysis. The final external interface capability is through a tape drive unit. This allows storage of all the information in the DAS memory, including the set-up of all the menus. This tape drive feature would be essential if the DAS 9100 mainframe was operated independently. This is because of the time it could save in the set-up of menus and in the generation of long sequences of patterns. The configuration at NPS does not have a tape recorder, but the DVS50 personal computer supported software performs the function of the tape recorder, so the tape recorder is not necessary.

An additional interface option available for the DAS 9100 is a power supply to apply voltage to the module pods and the DUT. This power supply is not installed at NPS and a Power Designs Model 3650-S serves as the system power supply.

C. 9100 DEVICE VERIFICATION SOFTWARE

1. Background

The most difficult aspect in using the DAS 9100 mainframe as a stand-alone system is the setting up the menus, data acquisition channels and pattern generation sequences via the DAS 9100 keyboard. The 9100 Device Verification Software (91DVS) allows the user to specify the data acquisition channels and generate long sequences of patterns through a high level language program. The 91DVS provides a user-friendly menu driven program to functionally test the DUT and is defined in the 91DVS user's manual [Ref. 20]. The 91DVS requirements include DOS 3.0 or higher, 512K-byte of memory, IBM color graphics card and a GPIB interface board supplied with the 91DVS software. Two different software packages are provided with the 91DVS, DVS50 and DVS25. The functions of both programs are similar, but the DVS50 is installed at NPS because it offers an automatic compare feature and supports the tri-state control features.

This section will include the capabilities and limitations of the DVS50 program, the menus required for operation, and a description of the two data files necessary to provide input to the DVS50.

2. DVS50 Capabilities and Limitations

The DVS50 program can support up to three DAS 9100 mainframes and Table 9 on page 41 lists the DAS 9100 modules that are supported by DVS50. While the DVS50 can support up to 192 pattern generation channels, all pattern generator modules must reside on the same DAS 9100 mainframe. The acquisition modules may be in any of the three DAS 9100 units but, if both the 91A08 and 91A32 modules are used, they must not be mixed in the same DAS 9100. Additionally, if a 91A08 module is used, one must be installed in slot six of the DAS 9100.

Table 9. DVS50 MODULE SUPPORT

MODULE	TYPE	SPEED
91S16	Pattern Generator	50MHz
91S32	Pattern Generator	25MHz
91A08	Data Acquisition	100MHz
91A32	Data Acquisition	25MHz

An additional limitation is that the clock rate for the pattern generator and the data acquisition modules must be clocked at the same rate. The acquisition module are clocked externally by the master clock of the 91S16 or the 91S32 module.

3. DVS50 Input Programs

There are two user generated data files necessary to set-up the DVS50 for testing: test program file, ".src", and the test pattern file, ".das". The test program file specifies the test sequence, timing, threshold and power supply definitions. Figure 13 on page 43 was the test program used for NPS CORN88, and it will be examined to illustrate the details of the ".src" program. The key word **PROGRAM** must start the ".src" program with the name of the program following this key word. The asterisk, "*", indicates that the remainder of the 80 column line is ignored. This is used to section the program and to provide comments within the program to increase clarity of the program. The next section is headed by the key word **PINDEF** and it identifies pin names, pin number assignments on the fabricated chip and the DVS50 attributes for that pin. The following attributes are supported by DVS50:

- PAT--pattern generator output channel
- TRI--tristable pattern generator output channel
- ACQ--data acquisition channel up to 25MHz

- FAST--data acquisition channel up to 100MHz
- CLK--pattern generator clock output
- PS--power supply output that is further defined in the **PSDEF** section

As a matter of course the "ACQ" attribute is attached to as many of the input channels as possible to monitor the input patterns. The end of each section is the key word **END**.

The **TIMEDF** section sets the pattern and acquisition clock. Remember, that in the DVS50 the two clocks must be equal, so the program allows only the "PAT" input and that can vary from 40ns to 5ms. The **THRESHOLD** section sets up the threshold voltage value. TTL is a predefined threshold voltage of +1.5V but variable voltages can be assigned ranging for -2v to 5v. Notice that the MOS option of -10.0V to +20.0V can not be used in the DVS50. The **PSDEF** section details the values in mV and mA for voltages and current respectively. The ground connection of the DUT must be defined as 0 mV. Figure 13 on page 43 does not include a **DATADef** section because the pattern sequences are defined in the ".das" file. All ".src" programs must end with **BEGIN**; and **ENDS** to control the DAS 9100 tester.

The ".das" file for NPS CORN88 is listed in Figure 14 on page 44. The first line is the name of the program. The second and third lines are any two integers. These lines are used in the test result program that will be discussed below. Line 4 specifies the number of pins that DVS50 needs to generate patterns for. The pin names are then listed in rows that correspond with the pattern sequence in the respective columns. For example, the input channel PADIN will always have a pattern specified by column one of the pattern sequence and CLO's pattern will be located in column 24. An EOF at the end of the program indicates the end of the pattern.

The test result file ".A01" is in the same format as the ".das" file except that only acquisition channels are reported. The clock rates for the pattern generator and data acquisition modules are included in lines two and three respectively. This file can be used to compare against another file transferred from the simulator output to validate the output vectors.

This method of designating patterns and data acquisition parameters is much easier and quicker than using the DAS 9100 keyboard. A translator program, MOS2DAS, was written to generate the ".das", ".src", and the ".sim" files from the results of the MOSSIM II simulation. This translator will be discussed in the next chapter.

```

PROGRAM chip;
PINDEF;
VCC      : 1,  PS 2;
OUT2     : 2,  ACQ;
OUT1     : 3,  ACQ;
PADIN    : 4,  PAT,ACQ;
PADOUT   : 5,  ACQ;
STSTOUT  : 6,  ACQ;
IN16     :10,  PAT,ACQ;
IN15     :11,  PAT,ACQ;
IN14     :12,  PAT,ACQ;
IN13     :13,  PAT,ACQ;
IN12     :14,  PAT,ACQ;
IN11     :15,  PAT,ACQ;
IN10     :16,  PAT,ACQ;
IN9      :17,  PAT,ACQ;
IN8      :18,  PAT,ACQ;
IN7      :19,  PAT,ACQ;
IN6      :20,  PAT,ACQ;
IN5      :21,  PAT,ACQ;
IN4      :22,  PAT,ACQ;
IN3      :23,  PAT,ACQ;
IN2      :24,  PAT,ACQ;
IN1      :25,  PAT,ACQ;
DATACON  :26,  PAT,ACQ;
REFCON   :27,  PAT,ACQ;
MASKCON  :28,  PAT,ACQ;
SPCON    :30,  PAT;
TSTCON   :31,  PAT,ACQ;
GND      :35,  PS 1;
BMSBOUT  :36,  ACQ;
OUTCON   :37,  PAT,ACQ;
OUT5     :38,  ACQ;
OUT4     :39,  ACQ;
OUT3     :40,  ACQ;
CLO      :29,  PAT,ACQ;
END;
TIMEDEF;
* Clockrate Pattern Generator.
PAT : ns 100;
END;
THRESHOLD;
* Threshold for Acquisition Module.
ACQ : TTL;
END;
PSDEF;
* Definition of Power Supply.
1 : mV 0;
2 : mV 5000, mA 3000;
END;
BEGIN;
END$

```

Figure 13. NPS CORN88 DVS50 .SRC File

```

CHIP
1
1
24
PADIN
IN16
IN15
IN14
IN13
IN12
IN11
IN10
IN9
IN8
IN7
IN6
IN5
IN4
IN3
IN2
IN1
DATACON
MASKCON
REFCON
TSTCON
SPCON
OUTCON
CLO
11111111100000000010100
11111111100000000010101
11111111100000000010100
11111111111111110100100
11111111111111110100101
11111111111111110100100
111111111000000011000100
111111111000000011000101
111111111000000011000100
111111111000000011000110
111111111000000011000111
111111111000000011000110
1111111110000000111000110
1111111110000000111000111
1111111110000000111000110
1111111110000000111000111
1111111110000000111000111
1111111110000000111000110

```

Figure 14. NPS CORN88 DVS50 .DAS File

4. DVS50 Menus

Figure 15 on page 46, modified from Ref 20,p. 3-2, illustrates the system flow, from verifying the configuration file, compiling the test program, to running the tests and displaying the results. This flow is controlled by the use of menu driven screen displays. The first menu that the user encounters is the **MAIN** menu contains entries into all the of the first level menus that are listed below:

- **HELP**
- **Compile Test Program**
- **Restore Test Program**
- **Information**
- **Enter Test Menu**
- **Update Configuration File**
- **Display Test Data**
- **Compare Data**
- **Quit**

The screen cursor is controlled by the arrow keys and a function or menu is called by positioning the cursor and pressing the "Enter" key. The **Update Configuration File** is to entered first if the user needs information about the configuration or needs to modify the current configuration due to a modification in the DAS 9100. This file mirrors the modules that are installed in the DAS 9100, power supply connections and controller specifications. This file is set-up to the DAS 9100 current configuration and usually does not need not be entered.

Usually, the user will select the **Compile Test Program** as the first entry in the DVS50 menu. This menu uses the ".das " and ".src" programs that were previously discussed and compiles a DVS control file and test pattern file to set-up the DAS 9100. This program also checks for errors in the ".das" and ".src" files and creates the channel specification list which describes the connections between the pattern generation and data acquisition pods and the DUT. Figure 16 on page 48 is the channel specification produced for NPS CORN88 and the hook up of this specification will be discussed latter. The **Restore Test Program** uses the test files that were previously compiled in the **Compile Test Program** menu.

After a test program file has been compiled the next step is to enter the **Information** menu. This menu allows the user to list or print the following: the test program,

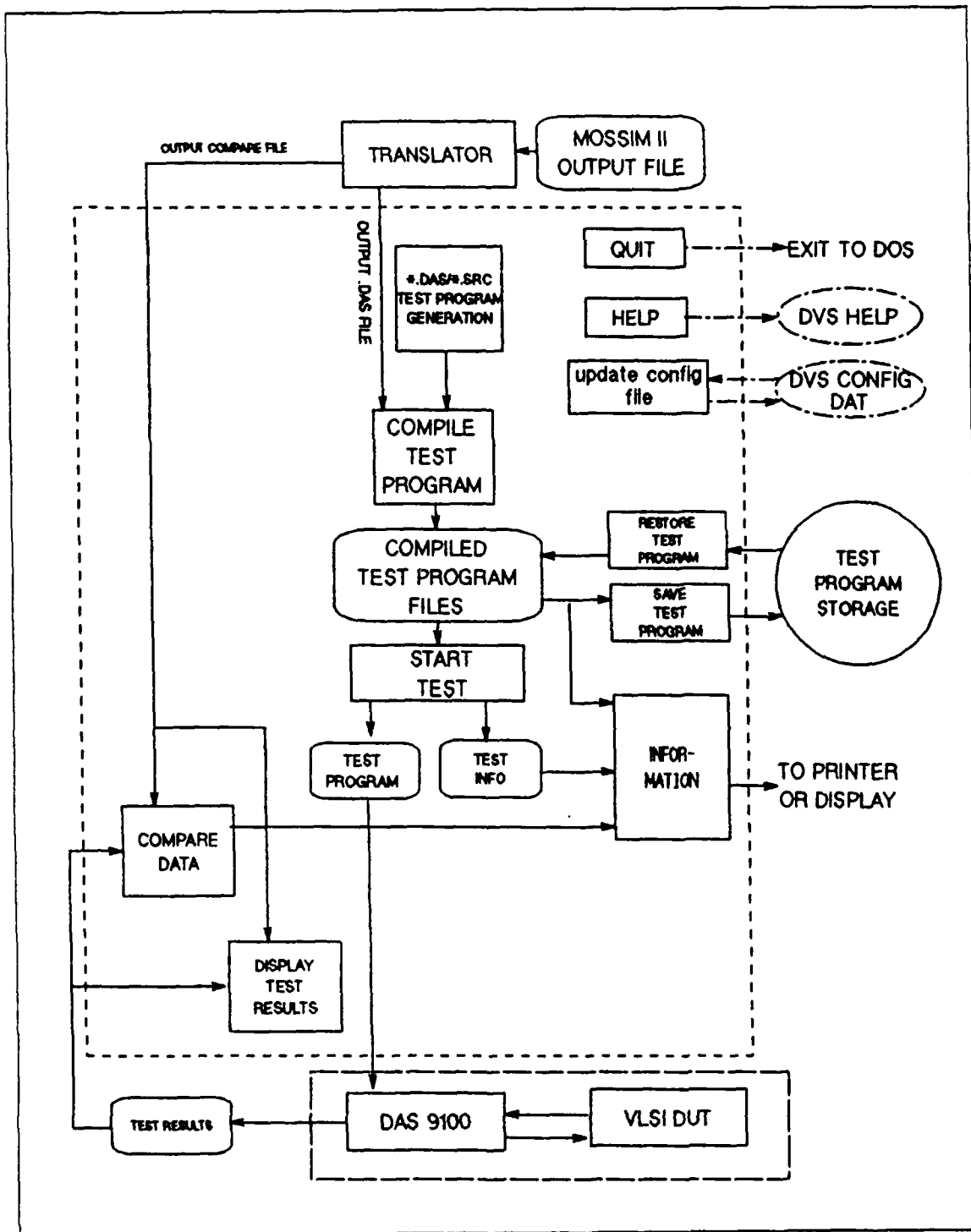


Figure 15. DVS50 Processing Flow

the test record and the compare data. The principle purpose of entering this menu is to obtain a listing of the channel specifications created during the compilation of the test programs.

Once the DAS probes have been connected to the DUT the **Enter Test Menu** is entered and "run test" is selected. This sets-up the DAS 9100, generates the sequence of patterns and the resulting acquired data from the DUT is uploaded to the PC. The "run test and compare" entry in this menu will run the test and compare the acquired data to a reference file and report differences. Also, within this menu is the "enter device commands menu". This sub-menu allows the user to send single commands to the devices supported by the 91DVS. This function is used for debugging purposes and for experienced users and is covered in detail in the DVS user's manual [Ref. 20, pp. 4-4,4-8].

The final DVS50 menu, and the most interesting to the user, is the **Display Test Result**. When entering this menu, the user is prompted for the names of two files, normally the recently acquired data file and a comparison file. Figure 18 on page 51 is an example of an observation window of the displayed file. A timing diagram of up to 24 channels for the two files is overlaid on the display screen. Function keys are used to expand and compress the screen, move the observation window left or right, and move the cursors left or right. The other useful features of this display are controlled by the command line. Channels can be listed, deleted, and inserted from this command line. A "scale" command sets the width of the observation window and the "time" command sets the beginning of the displayed window to the sequence specified. A "search" function, also available from the command line, specifies a particular bit pattern for which to search.

The primary disadvantage of this of this display is that a hard copy of the waveform timing diagram is unavailable, except through a print screen utility. This provides adequate results for analysis and demonstration but not enough resolution for inclusion in reports.

The 91DVS program greatly reduces the time spent in setting up the DAS 9100, generating pattern sequences, and displaying test results. The capabilities provided by the 91DVS are extensive enough for all but the most sophisticated user and it provides an excellent indoctrination into digital signal analysis.

5. NPS CORN88

a. Set-up

One aspect of testing that remains, regardless of whether the DAS 9100 or DVS50 techniques are used, is the connection of the DAS 9100 modules and probes to

```

*** CHANNEL SPECIFICATION LIST ***
ATTENTION : Connect the DAS-PODs to the pins of the DUT
              according to the following list.
              Take care - incorrect connections may cause
              permanent damage to the DAS-PODs or the DUT !
*****
* Trigger channel(s):          PG-POD          ACQ-POD          *
*                             1B-STB-1         5D7-1             *
*****
* ACQ clock channel(s):       PG-POD          EXTCLK-POD         *
*                             1B-CLK-1         7C-CLK1-1          *
*****
* NR: NAME    PINNUMBER      POD(S)
*              PG    ACQN    ACQF    other
*              PS2
* 1  VCC       31
* 2  OUT2      32          5D6-1
* 3  OUT1      33          5D5-1
* 4  PADIN     34    1B7-1  5D4-1
* 5  PADOUT    35          5D3-1
* 6  STSTOUT   36          5D2-1
* 7  IN16      40    1B6-1  5D1-1
* 8  IN15      1    1B5-1  5D0-1
* 9  IN14      2    1B4-1  5C7-1
* 10 IN13      3    1B3-1  5C6-1
* 11 IN12      4    1B2-1  5C5-1
* 12 IN11      5    1B1-1  5C4-1
* 13 IN10      6    1B0-1  5C3-1
* 14 IN9       7    1A7-1  5C2-1
* 15 IN8       8    1A6-1  5C1-1
* 16 IN7       9    1A5-1  5C0-1
* 17 IN6      10    1A4-1  5B7-1
* 18 IN5      11    1A3-1  5B6-1
* 19 IN4      12    1A2-1  5B5-1
* 20 IN3      13    1A1-1  5B4-1
* 21 IN2      14    1A0-1  5B3-1
* 22 IN1      15    2D7-1  5B2-1
* 23 DATACON  16    2D6-1  5B1-1
* 24 MASKCON  17    2D5-1  5B0-1
* 25 REFCON   18    2D4-1  5A7-1
* 26 SPCON    20    2D3-1
* 27 TSTCON   21    2D2-1  5A6-1
* 28 GND      25
* 29 BMSBOUT  26          5A5-1
* 30 OUTCON   27    2D1-1  5A4-1
* 31 OUT5     28          5A3-1
* 32 OUT4     29          5A2-1
* 33 OUT3     30          5A1-1
* 34 CLO      19    2D0-1  5A0-1

```

Figure 16. NPS CORN88 Channel Specifications

the pins on the DUT. Figure 16 on page 48 is the channel specification for NPS CORN88 and illustrates the power supply, pattern generation, and data acquisition DUT connections to be made.

The specified trigger channels for the pattern generator and data acquisition must be connected together to initialize and sync the two modules. The pattern generator module is connected to the external clock pod to provide the clocking rate for the DUT. To facilitate the connections of the numerous probes to the DUT, a test board, Figure 17 on page 50, was built. Connections for the pattern generator and data acquisition probes were provided to the DUT by quick release low insertion force connections. Additional connections were provided for power supplies and jumpers. The first connections to the test board are the grounds from the data acquisition pods and power and ground from the pattern generation pod. To keep line losses at a minimum the pattern generation probes have small amplifiers at the probe end called active pin drivers, or podlets. While this reduces line loss, it does generate heat at the podlet. To reduce heat build up in the podlet the power to the DUT should be left on for only the length of the test. Additional components are installed on the test jig to improve performance. The resistors eliminates possible podlet and DUT impedance mismatch. Gold plated connector pins are installed on the board to prevent corrosion within the podlet connector.

b. Results

The operational validation of NPS CORN88 was accomplished with the same reduced set of vectors that was used for design validation in MOSSIM II. Figure 18 on page 51 illustrates the results of the first test of the data register. Table 10 on page 52, test 1, sets the initial register conditions for this test. Notice that the resulting output is correct with a three clock phase propagation delay. The cursor marks the beginning of the correct output data. The '0' is then rippled down the data register bits to validate all the data input register bits, and the expected BCD value of 15 was observed at the outputs.

The second test sequence of vectors, where the output results should go from 16-0 to test the combination adder circuitry, are illustrated in Figure 19 on page 53 with the initial register values in test 2 of Table 10 on page 52. Notice that the output values for the 0 and 16 are the same, '00000'! This error occurred in all 12 fabricated chips and indicated that the OUT5 pin was always at a '0' state. To determine whether a fabrication error or a layout error was the cause of this malfunction, the MOSSIM II test sequence was run again on the final layout design of the chip. The test result for the

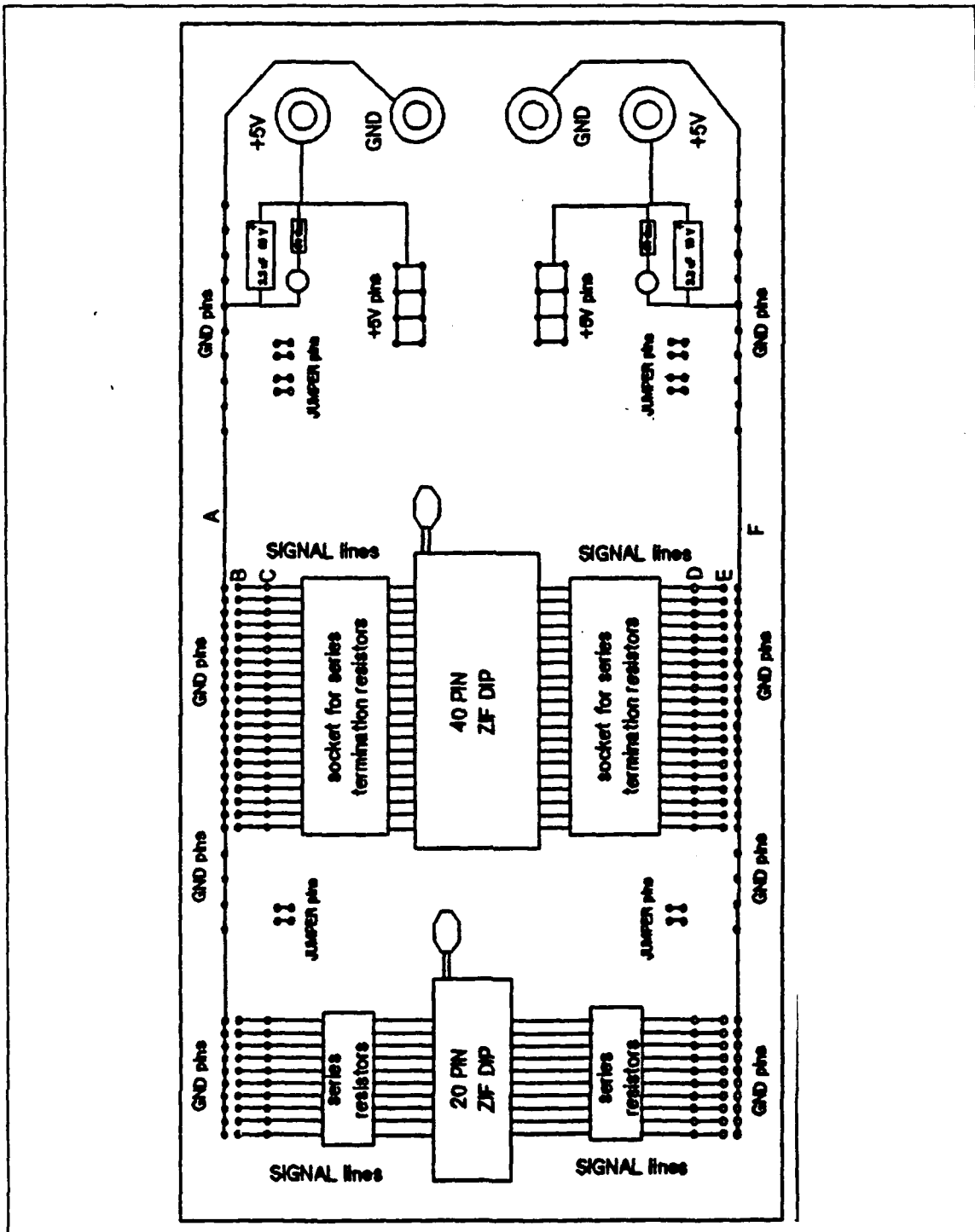


Figure 17. NPS Test Jig

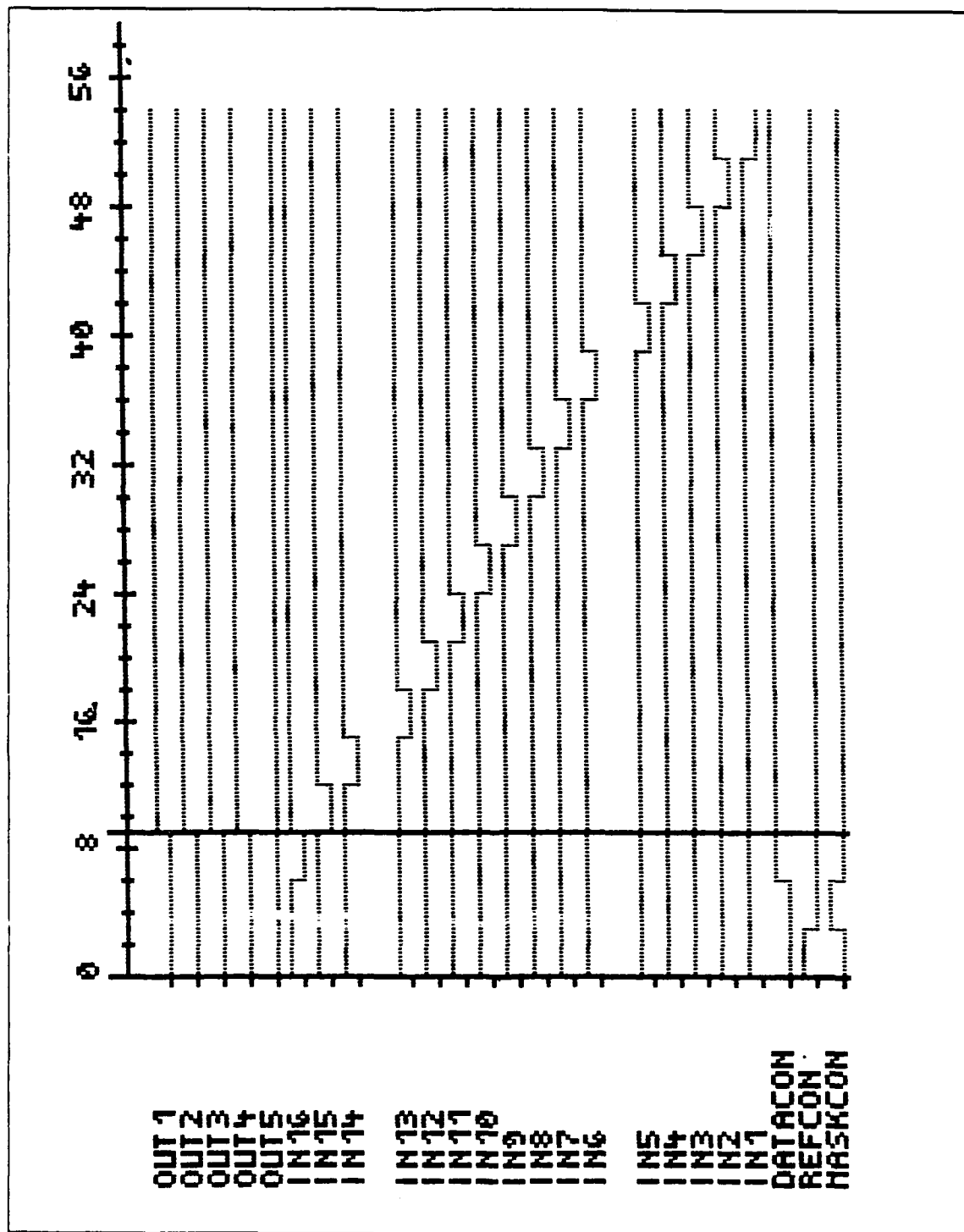


Figure 18. NPS CORN88 Data Register Test Results

Table 10. NPS CORN88 INITIAL REGISTER SET-UP

REGISTER	TEST 1	TEST 2
DATA REG IN	0111111111111111	1111111111111111
REF REG IN	1111111111111111	1111111111111111
MASK REG IN	1111111111111111	1111111111111111
XNOR REG OUT	0111111111111111	1111111111111111
BCD OUT, DIGITAL OUT	01111, 15	10000, 16

case of 16 matches resulted in the output 'X0000' vice '10000'. Investigating further revealed that Vdd was not connected to the final multiplexer controlling OUT5. After this layout was corrected, the output for 16 matches was '10000' indicating that it indeed was a layout problem not a fabrication problem that was discovered by the DAS 9100.

The third sequence of vectors that tested the reference bank by rippling an error down the reference bank is illustrated in Figure 20 on page 54, with the initial register values in Table 11 on page 56 test 3 values.

The masking tests started with no matches between the data and reference register as illustrated in test 4 of Table 11 on page 56. Figure 21 on page 55 demonstrates the effect of rippling two zeros, '00', through the mask register to obtain an output of '00010'. Additionally the odd values of masking combinations to further test the masking feature. The partitioning of the layout design was verified by observing the serial output STSTOUT after activating the testcon line and also observing the MSB line from the combiner adder.

The previous test sequences verified the various register, the xnor circuitry, and the combiner adder for parallel operations. An additional test sequence needed to be run to validate the serial input functions of the chip. A different test program was written to test these features. The rewriting of the test programs brought to light a limitation of this DAS 9100 configuration of only one 91A32 data acquisition module and one 91S32 pattern generator module installed. While it was relatively easy to generate a new ".das" and ".src" file, this changed the pod assignments to the DUT and the original connections had to be modified. This modification was required due to the limit on the number of data acquisition channels that could be acquired. The original input files used all 31 data acquisition channels including the monitoring of the 16 parallel data

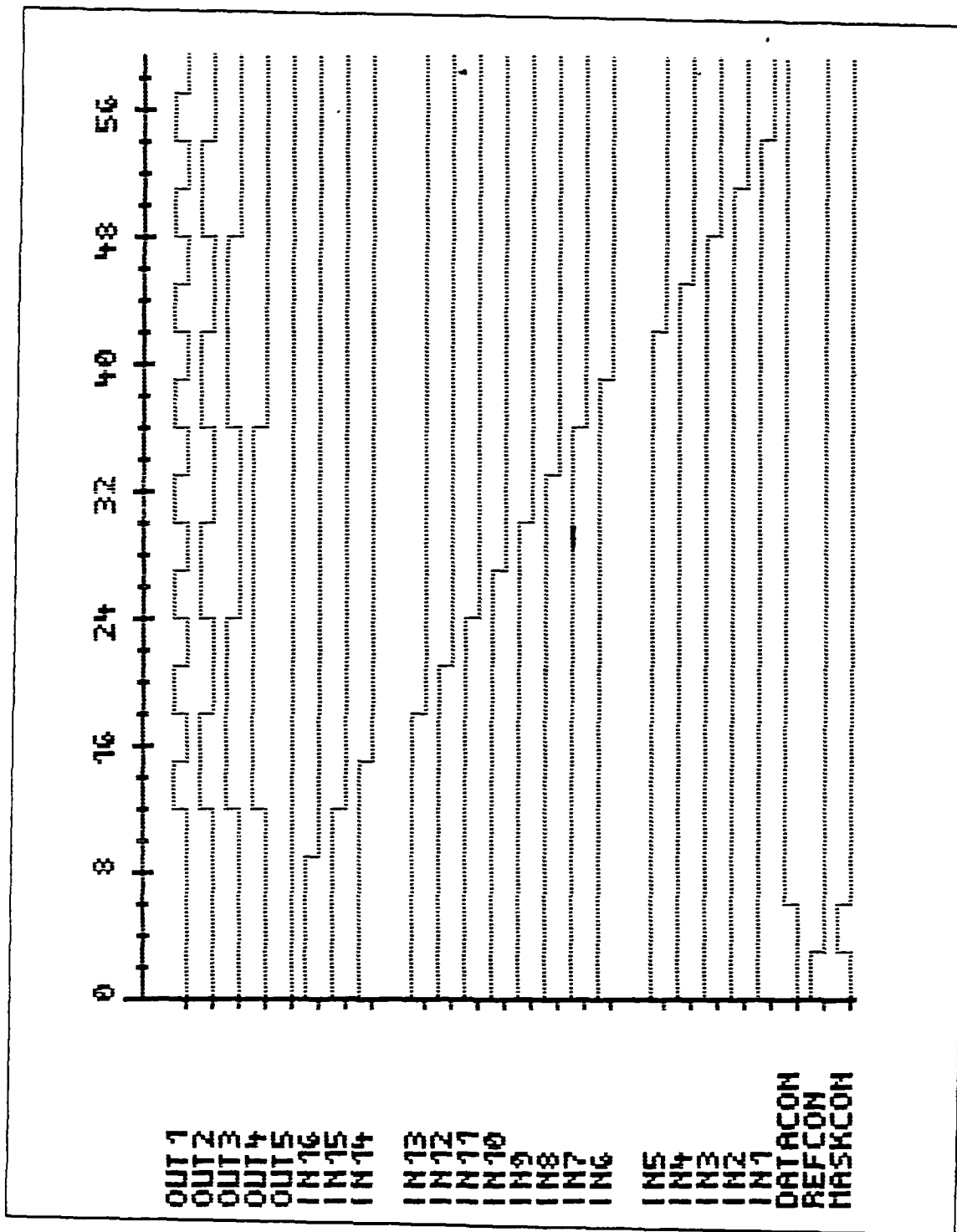


Figure 19. NPS CORN88 Combiner/Adder Test Results

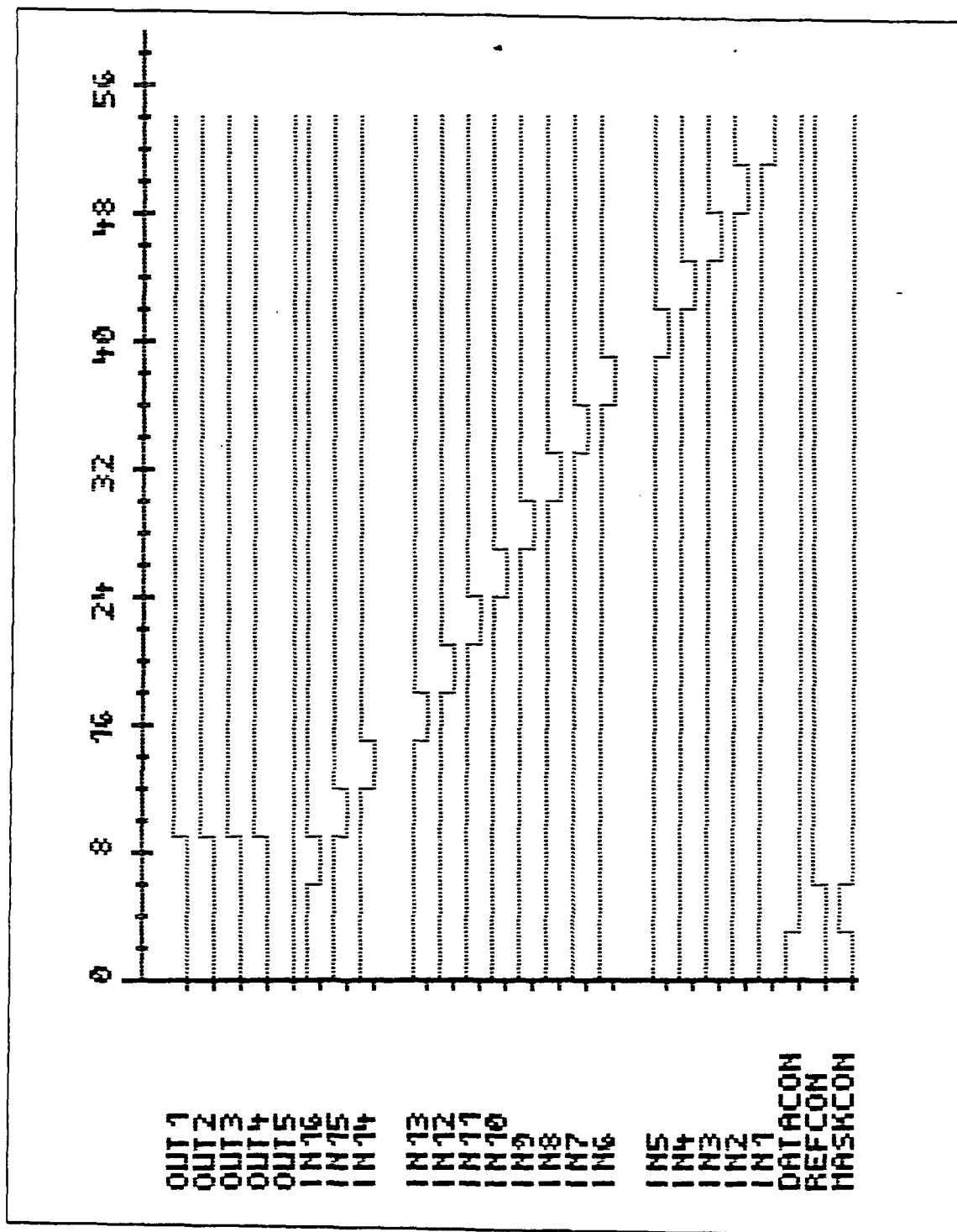


Figure 20. NPS CORN88 Reference Register Test Results

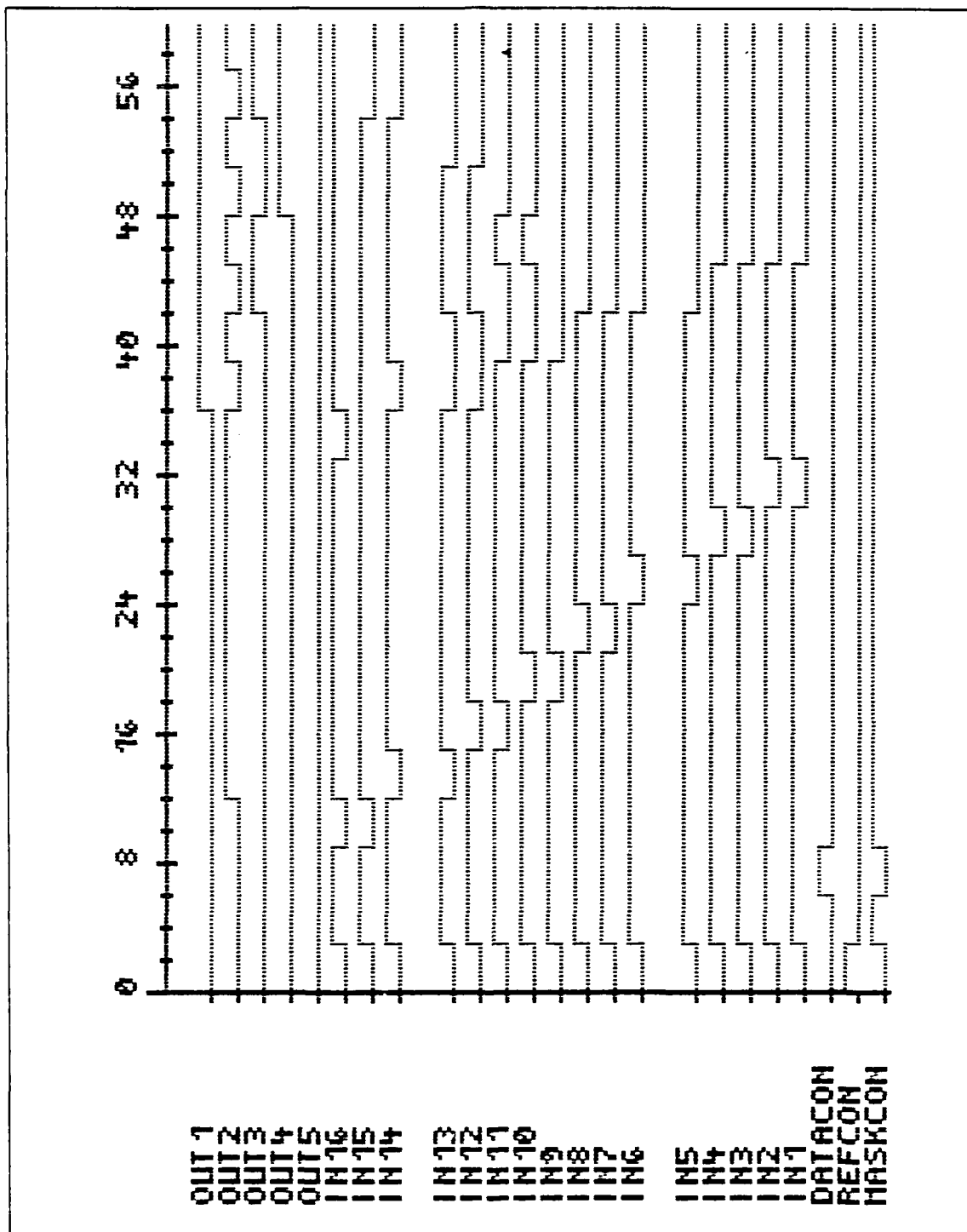


Figure 21. NPS CORN88 Masking Test Results

Table 11. NPS CORN88 REFERENCE AND MASKING INITIAL REGISTER VALUES

REGISTER	Test 3	Test 4
DATA REG IN	1111111111111111	1111111111111111
REF REG IN	0111111111111111	0000000000000000
MASK REG IN	1111111111111111	0011111111111111
XNOR REG OUT	0111111111111111	0000000000000000
BCD OUT DIGITAL OUT	01111/15	00010/2

input pins. The new program did not need to input a pattern or monitor the parallel input pins. Therefore, a new testing scheme had to be developed to generate patterns for the serial inputs and to monitor these serial inputs with data acquisition channels. While the modifications took only a short time, it would have been convenient to have an additional 91A32 installed so that the original program could have been written to support the testing of both parallel and serial data inputs. Figure 22 on page 57 illustrates the serial data results.

The test results validated the operational functionality of NPS CORN88. They also illustrated that the DAS 9100 series tester, in conjunction with the 91DVS and DVS50 software, was capable of providing the necessary stimulus to test a full custom VLSI chip.

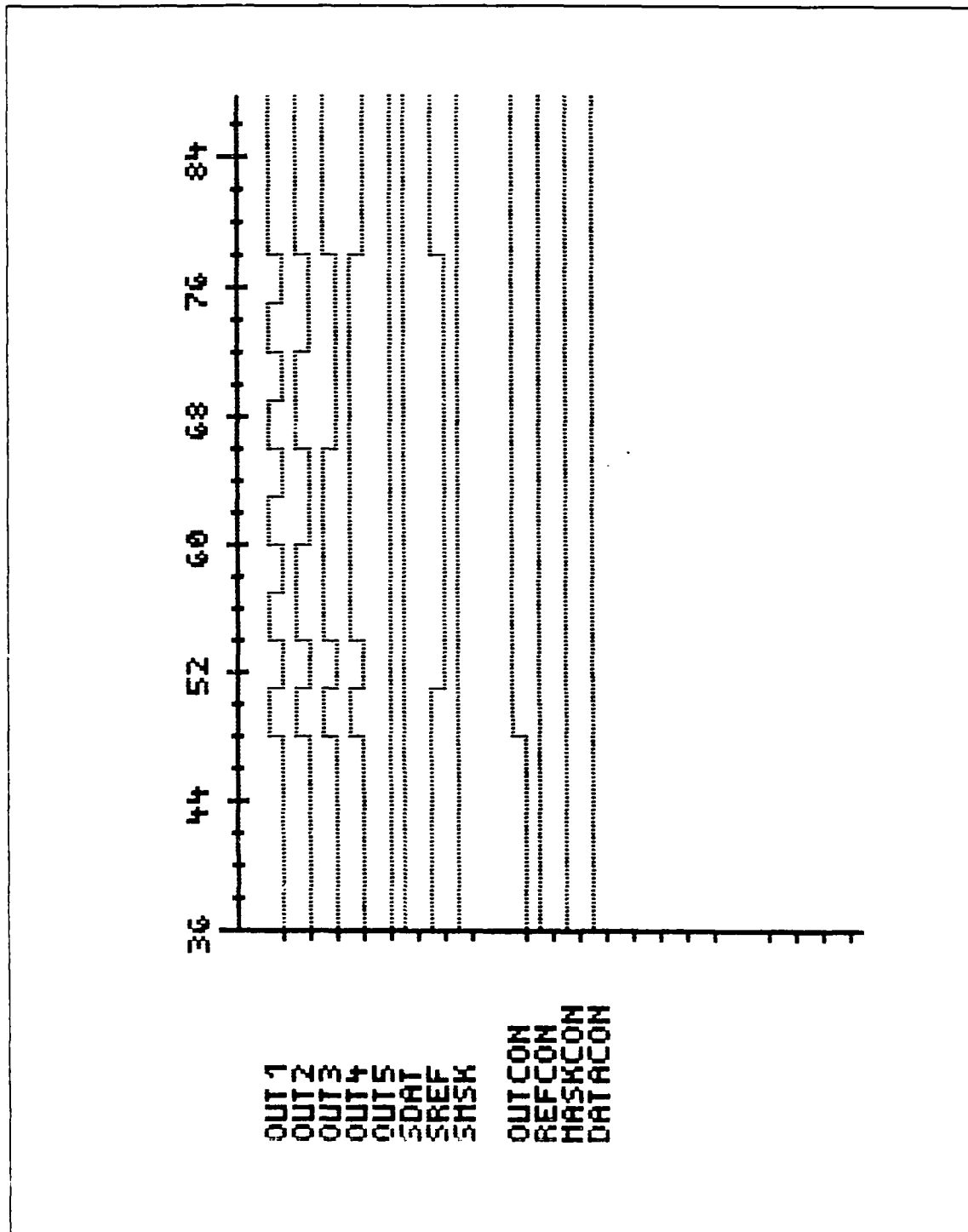


Figure 22. NPS CORN88 Serial Inpt Test Results

IV. MOS2DAS TRANSLATOR

A. BACKGROUND

The MOSSIM II simulation program functionally validates a circuit design by using test sequences to stimulate input pins and by monitoring the pins of interest. The test engineer must develop a set of vectors within this test sequence to validate the proper operation of the design. When the design has been validated and the chip has been manufactured, the test engineer must again establish a test program for the manufactured chip. With the NPS digital analysis system, which includes the DAS 9100 mainframe modular tester married to the DVS50 software, the test engineer inputs these test sequences through the DVS50 ".das", ".src", and ".sim" input files.

An interactive translator that converts the test vectors that were defined in the MOSSIM II simulation program to the DVS50 input files provides three major advantages. The most obvious advantage of a translator is that it saves the test engineer's time. MOSSIM II test sequences that were previously defined would be generated without the test engineer having to input the duplicate pattern sequences in DAS format. For a small test program the time savings would be insignificant, but as the test vector sequence scheme became longer and more complex, more time would be saved. The second benefit of using the translator is that the exact duplication of the MOSSIM II test pattern sequences and results are ensured. This is important because the results of the fabricated chip should mirror the results of the MOSSIM II simulation. The third advantage in using the translator is that the test engineer does not need to learn how to produce the three DVS50 files. This is only a minor consideration because the input files are relatively straight forward and easy to learn.

This chapter will discuss the capabilities and limitations of the MOS2DAS translator by examining the MOSSIM II commands that are supported by the translator. Additionally, the shell program that was developed to control the file manipulation involved in the translation will be examined. The source code for the shell program and the Fortran source code for the MOS2DAS translator are included in Appendix B.

B. MOS2DAS CAPABILITIES AND LIMITATIONS

1. MOSSIM II Commands

The MOS2DAS translator program generates three DVS50 input files from the MOSSIM II ".cpy" and ".ntk" files: the test program file, ".src", the test pattern file,

".das", and the test result file, ".sim". The ".cpy" file is used because it includes the outputs from the pins of interest. Several different conventions and commands can not be translated to the DVS50 input files because the DVS50 can not support them. When running the MOSSIM II program that the user wants to translate to DVS50 format, all MOSSIM II software switches must be set to the default values as listed below:

- ternary 0
- sharing 0
- static 0
- restored 0
- pseudo 0
- weaken 0
- redundant 1
- explain 0
- statistics 0
- echo 1
- case 1

Additionally, all watch commands must be modified to watch every phase, with the "watch *" command. This allows the program to report at the transition of every phase and more closely approximates what the DAS 9100 system will output.

Several MOSSIM II commands are used in the translation process, but some are not allowed. This is because the format of all the DVS50 input files requires that the pattern generation program must identify every channel prior to the first clock cycle. Therefore, all the pins and vectors of interest must be defined prior to the first cycle of the MOSSIM II program. Listed below are the MOSSIM II commands and their functions, as further defined in MOSSIM II user manual [Ref. 8, pp. 8-19] that are supported by the MOS2DAS translator:

- **CLOCK**--Defines the clocking scheme consisting of a set of clock nodes and a set of clock phases to be applied to these nodes. A MOSSIM II limitation is that, if more than one clock is defined, each clock must have the same number of phases in the sequence. For example, clock node ph1:0010 and clock node ph2:1000 are compatible non-overlapping clocks and are legal. The clock node ph1:010 and clock node ph2:0010 combination is not legal because the sequence length is not the same.
- **VECTOR**--This is used to assign one name to an ordered set of nodes. For example in Figure 23 on page 61 the vector ins contains all the parallel input nodes from in1-in16. A mixture of input and output nodes are not allowed in any one vector.

- **WATCH**--This command allows the user to identify nodes of interest to be printed out at each cycle command. To closely align the output of the MOSSIM II program and the results obtained by the DAS series from the manufactured chip the command "watch *" must be used to print the desired nodes after each phase of the clock.
- **SET**--Sets the node to a specific binary value. Used to generate the pattern generation sequence on that node or vector. Similar to the **FORCE** command except that the node or vector that is **SET** can be changed by the circuit operation to a new value.
- **FORCE**--Sets the node to a specific binary value. This node will always be that value, no matter what the circuit design tries to do to that node. The **UNFORCE** is not supported because the DVS50 program can not change node assignments after the first cycle command.
- **CYCLE**--This causes the network to run for a specified number of cycles. The translator supports up to 20 cycles per command.
- **QUIT**--This command indicates that the MOSSIM II simulation program is complete.

While this may seem to be a limited number of MOSSIM II commands, the test pattern generation sequence can be long and quite complex. NPS CORN88 was tested with these commands and they proved to be acceptable. The remaining key words that invoke MOSSIM II commands are not translated and the entire line that starts with a command that is not supported is disregarded. MOS2DAS can translate three states allowed by the MOSSIM II program at each node to the two states that are supported by the DVS50 software. The MOSSIM II states '0' and '1' are translated to DVS50 states '0' and '1', respectively, and the MOSSIM II 'X' state is translated to a '0'.

2. File Conversion Process

The MOS2DAS translator creates the ".das", ".src", and ".sim" DVS50 input files. The ".das" file is used to define the input pattern generation channels. It identifies the input pin or input vector by searching the ".ntk" file to determine whether it is an input or output pin. The first pin name after the command vector is checked and, if that pin is an input pin, then the vector is an input vector. The input array of pin names is developed from the vector assignment and the set and force commands. Remember, all input pins that are to be used in the DVS50 program must be defined prior to the first cycle statement. This is because at the first cycle statement the array of input pin names is sequentially listed to the ".das" file, with one pin name in each row. Figure 24 on page 62 is an example of a ".das" file. Each of the input pins is presented to the user by the interactive translator program. The user must be able to determine whether the input pin should be used for the translation and if the pin should also be monitored. The

```

>read CR_chip.ntk
#1
1386 nodes, 2939 transistors, 0 blocks
>
>initialize
>clock clk:010
>vector ins in1 in2 in3 in4 in5 in6 in7 in8 in9
           in10 in11 in12 in13 in14 in15 in16
>vector control datacon maskcon refcon tstcon
>vector misc spcon outcon
>vector outs out5 out4 out3 out2 out1
>vector out1 padout ststout bmsbout
>watch /* outs out1
>watch /* ins clk
>
>comment load reference data
>force ins:1111111111111111
>force control:1000
>force padin:1
>force misc:11
>cycle 1
1.1 | outs:00000 out1:100 ins:1111111111111111 clk:0
1.2 | outs:00000 out1:100 ins:1111111111111111 clk:1
1.3 | outs:00000 out1:100 ins:1111111111111111 clk:0

```

Figure 23. NPS CORN88 MOSSIM II Simulation Program

monitoring of the input pin will enable the user to display the input pattern during the display of the test results.

Once the pin names have been defined, the values assigned to each input pin name are determined by examining the force or set command for the vector or input pin defined. These values are also stored in an array of input values and, when the cycle command is recognized, the input pattern array is listed to the ".das" file as one row. The only remaining value to define in the ".das" file is the clock. The clock pins are handled by the DVS50 program as just another set of input pins. For each cycle of the network the translator produces the same number of input patterns as there are phases in the clock. For example, if there is a three phase clock, the ".das" file produces three rows of binary data for each cycle of the network. The pin names assigned to the clocking scheme are added to the input pin list directed to the ".das" file and the clock phase value is appended to the row of input values.


```

CHIP
1
1
24
PADIN
IN16
IN15
IN14
IN13
IN12
IN11
IN10
IN9
IN8
IN7
IN6
IN5
IN4
IN3
IN2
IN1
DATACON
MASKCON
REFCON
TSTCON
SPCON
OUTCON
CLO
11111111111111111111000110
11111111111111111111000111
11111111111111111111000110

```

Figure 24. MOS2DAS ".das" File

The remaining inputs to the ".das" file are the first four lines of the input file. The first row of the file is the program name and that is obtained from the interactive translator program. The second and third row of the ".das" file are any two integers. The fourth row is the number of pins in the input array in the ".das" file.

The ".sim" file is used to compare the output of the MOSSIM II simulator with the output from the DAS 9100 test system, ".A01". The pin names defined in the ".sim" file, as noted in the example of Figure 25 on page 64, are derived from the data acquisition assignments requested in the interactive translator program. Each of these pins in the ".src" program that have the attribute of "ACQ" assigned will be listed to the ".sim" file. When the pin names have been identified, the translator will search both the

input pin values using the set and force key words and the output line of the simulator. Each line of output will list a row of binary values to the ".sim" file. Except for lines two and three, the first four lines are similar to the ".das" file. Lines two and three in the ".sim" file are timing data that was obtained from the ".src" file.

One difficulty with producing a ".sim" file is that the MOSSIM II and DAS 9100 system sample the output data at different times. DAS 9100 samples the data at the leading edge of the pattern generator clock, and MOSSIM II samples the data after a phase is complete. This sometimes results in a one phase delay for the output to be observed by the DAS 9100. Figure 26 on page 65 illustrates the sampling differences between the two programs and demonstrates the need to implement a one phase delay of the MOSSIM II output to align the data. NPS CORN88 is an example of this output misalignment. To properly align the two output files the MOS2DAS translator is able to install a one phase delay in reporting the MOSSIM II output. During the interactive program the user will be asked if a phase delay is required, with the default value of a delay inserted. If, on comparison of the outputs, it is determined that a phase delay is not required, then the user will run the MOS2DAS translator again asking for no delay.

While the MOSSIM II program will identify syntax violations the MOS2DAS translation program will verify operational limitations imposed by the DAS 9100 module set up. The current configuration allows 31 input pins and the ability to acquire data on 31 pins of interest. This is because one channel from the data acquisition module is connected to a channel on the pattern generation module to facilitate synchronous timing between the two modules. The MOS2DAS translator will check the requested input, "PAT", and output, "ACQ", pin names prior to listing the pin names to the files and verify that the maximum number of input and output pins have not been exceeded. If the input or output pin count exceeds the limit, the user is prompted with an error message from the translator. This feature of the MOS2DAS program is easily modified when the module configuration of the DAS 9100 tester changes.

As mentioned earlier, the ".src" test program file supplies the DVS50 program with the pin assignments, assigns an attribute to each pin identified, sets up the definition of clock rates, determines the voltage threshold value for the circuit, and indicates the specifications for the power supply. This ".src" input file, Figure 27 on page 66, is generated by an interactive part of the MOS2DAS translator program. The program lists all the pins defined in the MOSSIM II simulation program and asks the user to pick an attribute and list the actual pin number on the fabricated chip. The output pins that were identified in the MOSSIM II program will automatically be assigned the attribute

```

CHIP
    100
    100

31
OUT2
OUT1
PADIN
PADOUT
STSTOUT
IN16
IN15
IN14
IN13
IN12
IN11
IN10
IN9
IN8
IN7
IN6
IN5
IN4
IN3
IN2
IN1
DATACON
REFCON
MASKCON
TSTCON
BMSBOUT
OUTCON
OUT5
OUT4
OUT3
CLO
00110111111111111111111110001100
011101111111111111111111100101101
011101111111111111111111100101100

```

Figure 25. MOS2DAS ".sim" File

of "ACQ", if monitoring is desired, and the input pins will be assigned "PAT" with an option of also assigning the attribute of "ACQ" to these pins. It is wise to monitor each pattern generator input line to display the pattern being generated. The power supply pins are not included in the MOSSIM II program and the user will be prompted to provide the names, pin assignment and supply voltages. All other inputs required from the user are requested for interactively with the valid options provided. When the ".src"

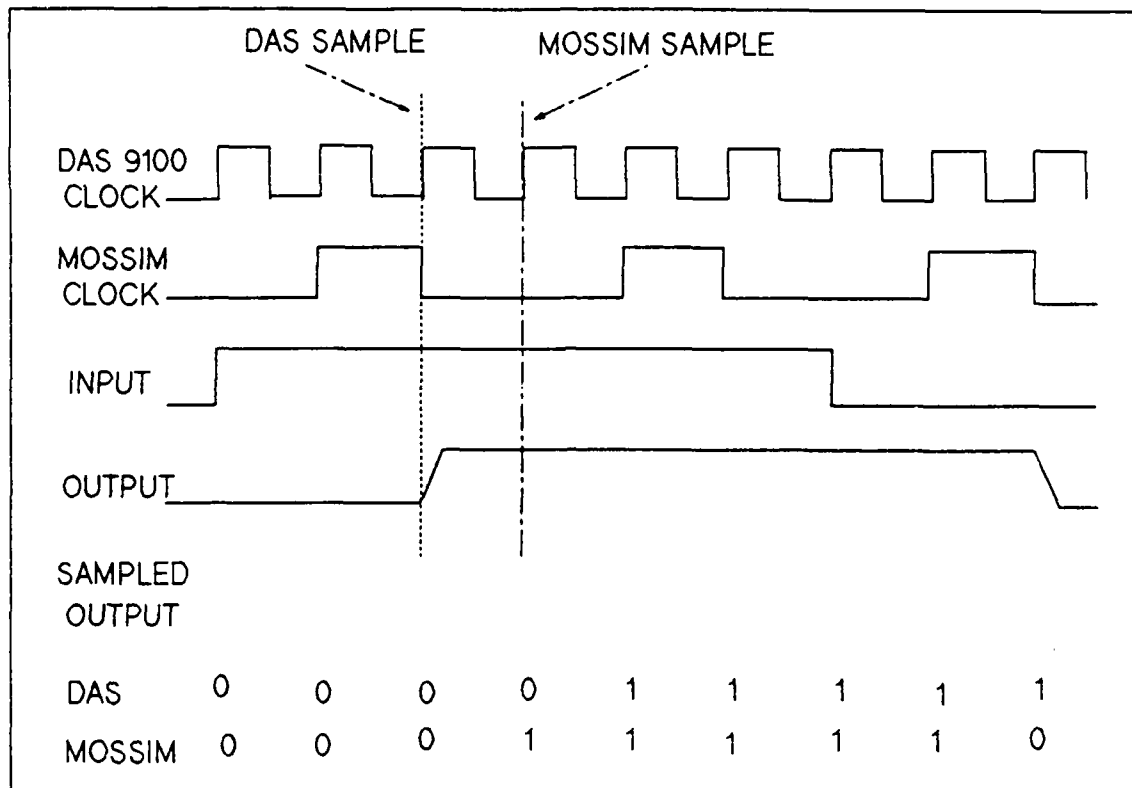


Figure 26. DAS 9100 and MOSSIM Sampling Differences

and ".das" input files are compiled by the DVS50 program a channel specification listing will be generated to define the connections between the DAS 9100 and the DUT. The DVS50 has the ability to control tri-state channels but that capability is not supported by the MOS2DAS translator. For further information on the tri-state function of the DVS50 consult the 91DVS user manual [Ref. 20, pp. 7-1, 7-4].

C. OPERATIONAL SHELL

A UNIX C-Shell program, "MOS2DAS", was written to control the interactive file manipulation required by the translator. This executable file, along with the "mostral" Fortran program, is located in the UW_TOOLS directory and is executable by any user that has access to the MOSSIM II simulator. This UNIX C-Shell program, Figure 28 on page 68, uses several UNIX utilities to manipulate the files: awk, grep, and diff. The user invokes the MOS2DAS translator by typing the following command at the UNIX prompt: mos2das < .ntk> < .cpy>.

```

PROGRAM chip;
PINDEF;
VCC      : 1,  PS 2;
OUT2     : 2,  ACQ;
OUT1     : 3,  ACQ;
PADIN    : 4,  PAT,ACQ;
PADOUT   : 5,  ACQ;
STSTOUT  : 6,  ACQ;
IN16     :10,  PAT,ACQ;
IN15     :11,  PAT,ACQ;
IN14     :12,  PAT,ACQ;
IN13     :13,  PAT,ACQ;
IN12     :14,  PAT,ACQ;
IN11     :15,  PAT,ACQ;
IN10     :16,  PAT,ACQ;
IN9      :17,  PAT,ACQ;
IN8      :18,  PAT,ACQ;
IN7      :19,  PAT,ACQ;
IN6      :20,  PAT,ACQ;
IN5      :21,  PAT,ACQ;
IN4      :22,  PAT,ACQ;
IN3      :23,  PAT,ACQ;
IN2      :24,  PAT,ACQ;
IN1      :25,  PAT,ACQ;
DATACON  :26,  PAT,ACQ;
REFCON   :27,  PAT,ACQ;
MASKCON  :28,  PAT,ACQ;
SPCON    :30,  PAT;
TSTCON   :31,  PAT,ACQ;
GND      :35,  PS 1;
BMSBOUT  :36,  ACQ;
OUTCON   :37,  PAT,ACQ;
OUT5     :38,  ACQ;
OUT4     :39,  ACQ;
OUT3     :40,  ACQ;
CLO      :29,  PAT,ACQ;
END;
TIMEDEF;
* Clockrate Pattern Generator.
PAT : ns 100;
END;
THRESHOLD;
* Threshold for Acquisition Module.
ACQ : TTL;
END;
PSDEF;
* Definition of Power Supply.
1 : mV 0;
2 : mV 5000, mA 3000;
END;
BEGIN;
END$

```

Figure 27. MOS2DAS ".src" File

```

# this is the beginning of all C Shell programs
# this checks if there are two arguments
if( $#argv != 2)then
    echo "usage: ntk2das <file.ntk> <file.cpy>"
    exit
endif

set ntkfile=$1
echo $ntkfile:r
echo $ntkfile:r.cpy
#check to see if the input file is a valid ntk file
if(! -e $ntkfile ) then
    set ntkfile=$1.ntk
    if( ! -e $ntkfile)then
        echo "cannot find the file $1 or $1.ntk"
        exit
    endif
endif

#check to see if the input file is a valid mos copy file

set mosfile=$2
if(! -e $mosfile ) then
    set mosfile=$1.cpy
    if( ! -e $mosfile)then
        echo "cannot find the file $1 or $1.cpy"
        exit
    endif
endif

# convert the files to be used for the translator
egrep ^i $ntkfile | awk '{print $2 }' > ntkin.dat
egrep ^s $ntkfile >pout1
egrep _ $ntkfile >pout2
diff pout2 pout1 > pout2
egrep '> s' pout2 | awk '{print $4}' > ntkout.dat
mostrat
cp mos.das $ntkfile:r.das
cp mos.src $ntkfile:r.src
cp mos.cmp $ntkfile:r.sim
rm mos.das mos.src mos.sim
rm pout1 pout2 ntkin.dat ntkout.dat

```

Figure 28. MOS2DAS C-Shell Operating Program

The C-shell program then checks to insure that these files are valid files in the directory and then renames the ".ntk" file to "transin.dat" for input to the "mostrat"

translator. These utilities search the ".ntk" file that has been defined by the user and set up a file of input pin names and a file of output pin names. These file names are "ntkin.dat" and "ntkout.dat" respectively and are used by the "mostral" Fortran program as input files. Now that the input files have been specified for the "mostral" translator that executable program is called. The "mostral" program lists the ".das", ".src", and "sim" files into the temporary files "mos.das", "mos.src", and "mos.sim" respectively. These files are then copied to the original file name with the new file extension of ".das", ".src", and ".sim". The temporary files are then erased and the translation process is complete.

V. CONCLUSIONS

A. SUMMARY

This thesis validated the engineering methodology to produce a full custom, fully tested, VLSI chip from an original design using Naval Postgraduate School assets. During this validation process several elements of the design and testing phases were noteworthy.

The first, and most important, element is the relationship between the design engineer and the test engineer. These functions, while at NPS, are often performed by the same individual, but, in the testing of NPS CORN88, the circuit design engineer and the testing engineer were not the same individual. This separation of responsibilities more closely duplicates industry's design and implementation of a full custom chip. When the responsibilities for producing a functional chip are shared, these key individuals must be involved from the initial design phase through the testing phase. Operating parameters and specifications must be defined for both the circuit design engineer and the test engineer. NPS CORN88 was a completed functional design with only an overview of the functions and capabilities provided. This made the testing strategy for the design difficult and could have lead to the development of invalid test routines.

Two MOSSIM II test routines were developed for NPS CORN88 at the functional design validation level. One test vector sequence produced 2^{16} 16-bit vectors, to duplicate all possible input values, and the second test sequence produce a greatly reduced subset of those vectors. Neither of these tests identified any problem with the circuit design. When the fabricated chip was tested with the DAS 9100, an error in an output pin was discovered. This reduced set of vectors was again run on the final design that was sent to fabrication. The error that appeared in the fabricated chip was duplicated in the circuit design simulation. This could only occur if some modification of NPS CORN88 has been done after the "final" testing. It is obvious that final testing must be accomplished after the final design is complete and just prior to sending the design mask to fabrication.

The concept of partitioning the design to allow for additional observability within the circuit also proved useful. When the error was identified during the testing of the fabricated chip, the fault was isolated to the final adder section. This was done by observing the outputs of the test register and the most significant bit from the combiner.

This was useful in the subsequent localizing of the error in the circuit design. While it is important to increase the observability of a VLSI chip, the primary limitation that is now associated with chip development is the limited number of pins in the fabrication package. When a standard frame package is used for fabrication, it is incumbent on the circuit designer and test engineer to utilize any extra pins to improve observability.

The advantage of using MOSIS to produce a limited number of chips for a new circuit design is also obvious. Being able to produce a small number of VLSI chips to verify the design and fabrication process greatly reduces the financial and schedule risk of large scale production. For a small expense, \$1400, and a modest six week fabrication schedule, twelve prototype VLSI chips were produced. The error that was discovered during the testing of these fabricated chips would have been very costly if this had been a large full scale production.

The engineering methodology for the design and production of a full custom VLSI chip is well defined and adequate, but it requires knowledge of a CAD tool, MAGIC, the MOSSIM II simulator and the interface between the two major components. The flexibility that is gained by using the MAGIC layout editor and various simulator models is over shadowed by a silicon compilation system that controls the entire design and testing process.

The DAS 9100 series tester as a stand-alone system is a capable and reliable digital analysis tool. A broad-based knowledge of the capabilities and testing procedures are required before even the simplest test is performed. Another limitation of this system revolves around the keyboard entries that are required to set up the system and to store pattern sequences. This limitation would be somewhat overcome if the tape recorder option was purchased to store this data so that the user would not have to reenter it after every power down of the equipment. However, when the DAS 9100 series tester is married to the 91DVS, and the DVS50 program the system is user friendly and easy to use for even the most inexperienced test engineer. This digital analysis package is capable of generating comprehensive test pattern vectors and displaying the results for analysis or comparison. The development of a translator, MOS2DAS, to convert MOSSIM II test vectors to the two programs needed by the DVS50 makes the test engineer job more efficient.

B. RECOMMENDATIONS

The current configuration of the DAS 9100 supports up to 32 data acquisition channels and 48 channels for pattern generation. This adequately tests a 40 pin VLSI

chip. An additional 91A32 data acquisition module should be purchased (\$5,480) to improve the VLSI data acquisition to up to 64 pins. If the DAS 9100 is envisioned to operate as a stand-alone tester, the addition of a tape recorder module would be considered essential. If the DAS 9100 continues to operate with the DVS50 software the addition of a tape recorder is not necessary. The current power supply configuration for the DAS 9100 only supports one set of voltages. Because there are many power supplies at NPS that could easily be added to the DAS 9100 system the purchase of the power supply module is not recommended.

Additional suggested research topics or projects are providing a translator from the GENESIL silicon compiler to DVS50 user programs and modifying the DVS50 software to allow the user to print a graphic display of digital waveform results. The current method is to capture and print the screen with a utility and that provides only moderate resolution.

C. BENEFITS

The benefits of this research were four-fold. A very thorough understanding and appreciation of the methodology for producing a custom VLSI chip with the MAGIC layout was gained and presented to the reader. The digital test facilities at NPS were examined, discussed, and a testing methodology was presented. Being able to follow a MAGIC circuit design, NPS CORN88, from design to testing of the fabricated chip was truly gratifying.

This thesis developed two tools that are related to the NPS digital analysis test facility. The MOS2DAS translator will allow the digital test engineer to concentrate on the development of MOSSIM II test vectors. The vectors that functionally validate the circuit design can be translated to the DVS50 input format, saving the test engineer time in testing the fabricated chip. Finally, the new user will find an introduction to the NPS digital test facilities in the tutorial provided in Appendix A.

APPENDIX A. NPS DIGITAL TEST FACILITY TUTORIAL

A. PURPOSE

The purpose of this tutorial is to provide an examination of the digital test facilities (DTF) at the Naval Postgraduate School (NPS). These test facilities at NPS include a digital analysis system, Tektronix DAS 9100 series tester, controlled by the device verification software (DVS50) installed on a personal computer (PC). While the DAS 9100 series tester can be operated as a stand-alone tester, the NPS configuration includes the link to the DVS50 software, and that will be the configuration discussed in this tutorial.

This tutorial will provide an overview of the capabilities and limitations of the digital test facilities. Additionally, it will use the arithmetic logic unit (ALU181) as a vehicle to follow the testing methodology. Upon completion of this tutorial the user will be able to generate the input files for the DVS50, set up the device under test (DUT), and navigate through the menus provided by the DVS50 program to compile and run a test, and to display the test results.

B. CAPABILITIES AND LIMITATIONS

As briefly discussed in the preceding section, the DTF is composed of many units. Each of these components provide some capabilities as well as some limitations. The DTF is comprised of the Tektronix DAS 9100 series tester, DVS50 software on a PC host, a power supply, and a test jig. The heart of the system is the DAS 9100 which performs the functions of data acquisition from and pattern generation to a DUT. The capabilities of the DAS 9100 are flexible, based on the modules that are installed in the DAS mainframe. The current configuration can support 32 channels of data acquisition and 48 channels of pattern generation. The installed modules also set the timing capabilities. The current configuration can support a timing rate from 40nS to 5mS or a maximum rate of 25MHz. The DAS 9100 sets up the connectivity to the DUT and controls the clocking of the system.

Equally important to the DTF is the DVS50 software, hosted on a personal computer collocated with the DAS 9100. This software is menu driven and uses the file generation file, ".src", and the test pattern file, ".das", to control the DAS 9100. If the heart of the DTF is the DAS 9100, then the brains of the system is the DVS50 software. The two components are connected with a general purpose interface board. The limitation of using the DVS50 software is the pattern generation clock and the data

acquisition clock are tied together and can not be run at separate speeds. Additionally, this clocking scheme takes one data acquisition channel and one pattern generation channel, so the current system is limited to 31 channels of data acquisition and 47 channels of pattern generation.

The power supply provides for 5 volts at 3 amperes. This restricts the current configuration to TTL logic levels, but the system can easily be configured to handle additional power supplies if necessary. The test jig was designed and manufactured to make connecting the module probes to the chip easier. The test jig can handle up to a 40 pin chip, but the test jig could be expanded. In summary, the DTF offers an excellent tool to test digital chips and the capabilities and limitations are summarized below.

- 31 DATA ACQUISITION CHANNELS
- 47 PATTERN GENERATION CHANNELS
- 40nS-5mS CLOCK RATE
- TTL OR ECL LOGIC LEVEL
- POWER SUPPLY ONLY 5 VOLTS AND 3 AMPERES
- TEST JIG SUPPORTS UP TO 40 PINS

C. DVS50 INPUT FILES

The two files that are required by the DVS50 software are the ".src" and the ".das" files. The ".src" file generates the configuration for the DAS 9100 and the ".das" file contains a description of the test sequence pattern to be generated. The arithmetic logic unit model 181, ALU181, will be used to demonstrate the functions of each of these files. The ALU181 performs 16 binary operations on two 4-bit words. Figure 29 on page 74 was assembled from a TTL handbook [Ref. 21] and the addition of two 4-bit words will be the operation implemented.

The ".src" file generates the configuration to the DAS 9100 and the ".das" file generates the pattern to the DAS 9100. A third file that can be used by the DVS50 program is the ".sim" file that can be used to compare the test results. These files must be created prior to entering the DVS50 test menu. This can be done in three ways, as outlined in Figure 30 on page 75. The first way is to generate the files on an editor the user is familiar with and copy that file from a floppy disk to the working directory, c: dvstest chip. The second method of producing a file is to create it on the VAX and use the NFT transfer method between the VAX and the PC. The third method is to access the editor after being in the working directory. The two editors that are available to that working

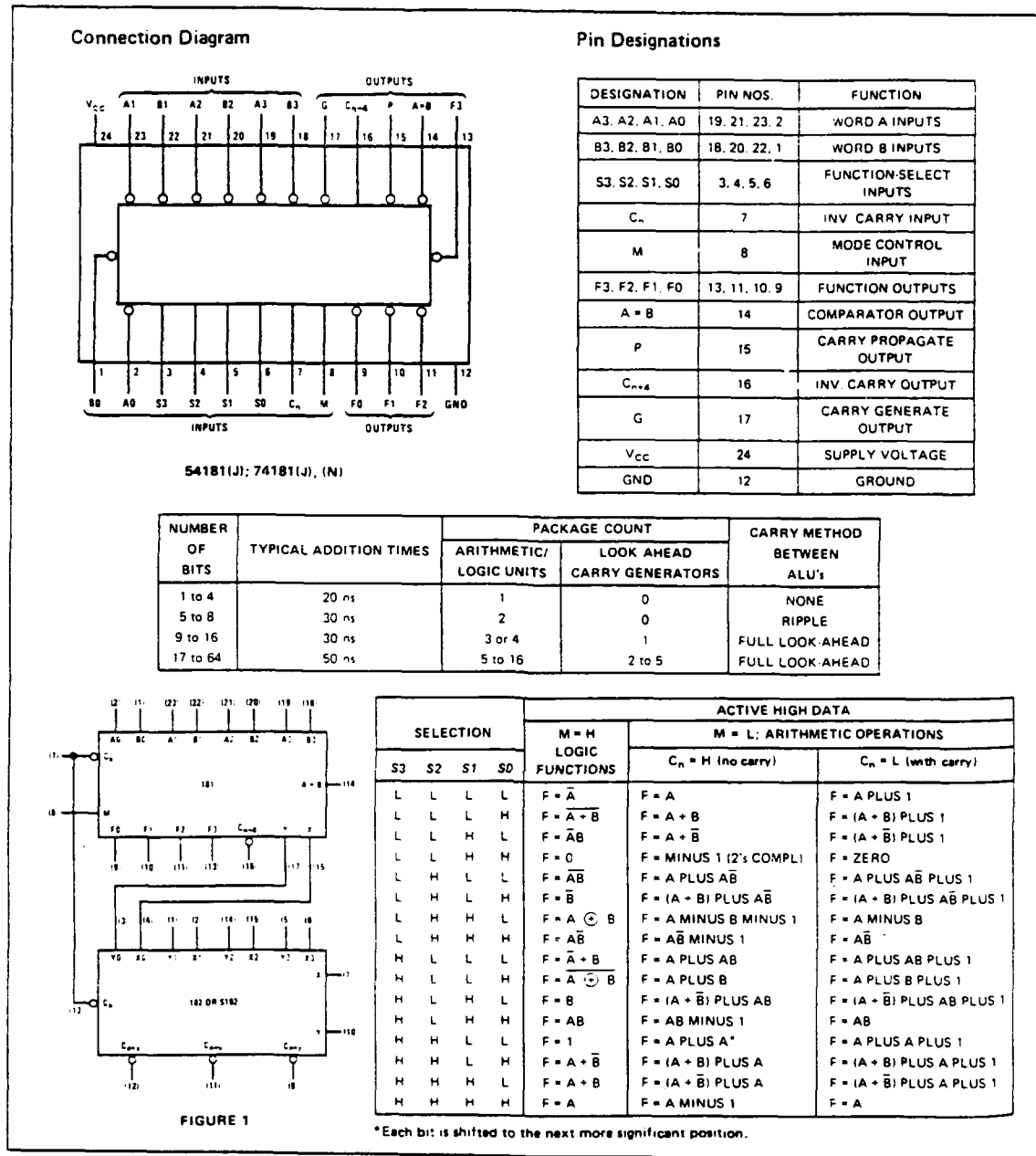


Figure 29. ALU181 Specifications

directory are KEDIT and EDLIN. Whatever method is used to generate the files, they must be located on the working directory prior to entering the DVS50 program and they should be removed from the working directory after the testing is complete.

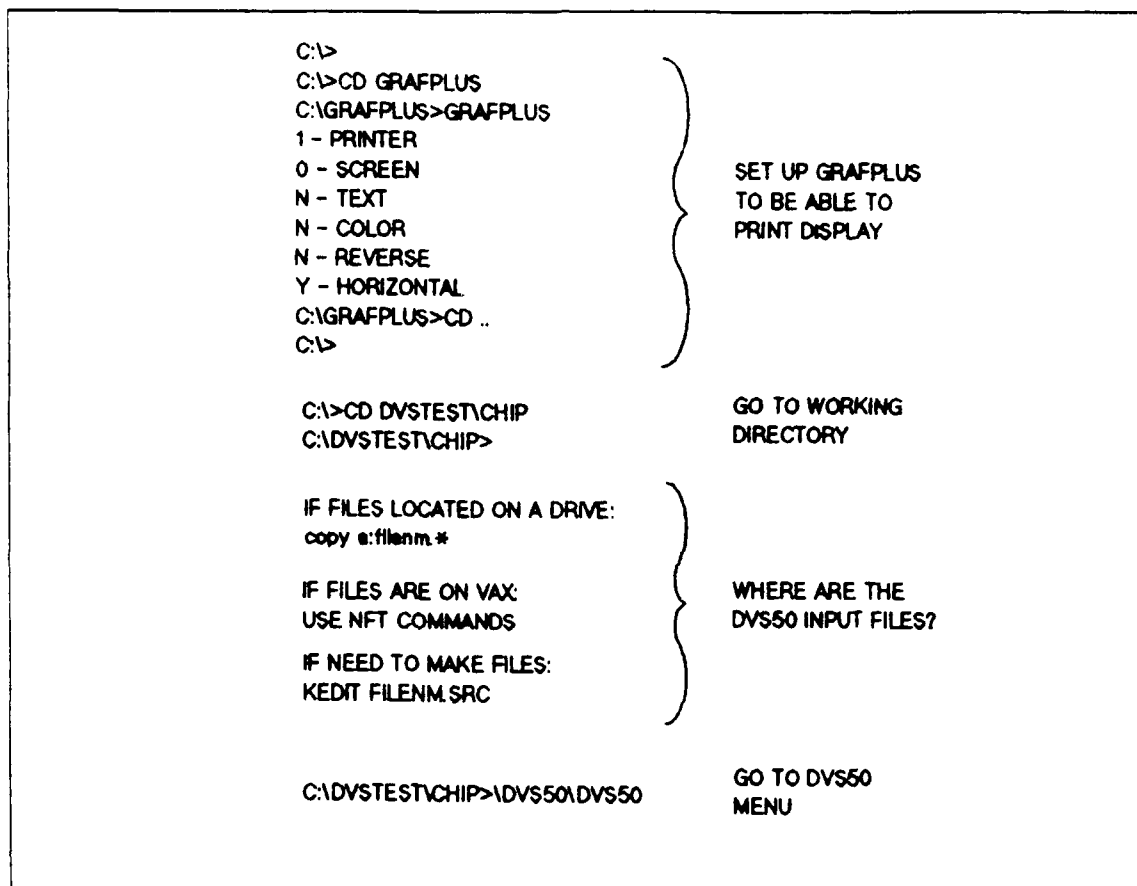


Figure 30. DOS Commands and File Directory Flow

The ".src" file that was used for testing the ALU181 chip is illustrated in Figure 31 on page 77. This file will identify the program name, pin names, pin number assignments, attribute assignments to the pins, clocking rates, power supply requirements, and required logic level. The illustrated file is straight forward, but some comments are in order to clarify some of the options. The attributes that can be assigned to the pin, for this example, are either the "PAT", "ACQ", or "PS". The "PAT" attribute identifies that the pin will be an input pin and a pattern channel is reserved by that name. The "ACQ" attribute delineates a data acquisition channel. It is important to monitor, acquire, pattern generation channels to observe these channels on the displayed results. The "PS" attribute indicates that the pin is a power supply that will be defined latter in the "PSDEF" section. The other sections are self explanatory except for noting that the

power supply parameters must be given in millivolts and milliamps. The ground must be defined as 0 mV for this section to be complete.

The ".das" file defines the test sequence pattern to be generated on the input pins. Figure 32 on page 78 illustrates the ".das" file that generated the patterns sequence for the ALU181 chip. The first line identifies the name of the chip and the next two lines can be any two integers. The fourth line indicates the number of input pins, and that is followed by each row containing a pin name. Each pin name corresponds to a pattern column that appears after the last pin name. For example, the first pin name is "B3" and the binary value in column one of the test sequence is assigned to that pin name. The last pin name will correspond to the last column in the test vector sequence. Values for the test sequences were developed to test several combinations of input values and to force the various combinations of output results.

The ".sim" file in Figure 33 on page 79 is similar to the ".das" file but it contains all the pin names that had the attribute of "ACQ" in the ".src" file. The only difference is lines two and three which indicate the clocking rate of the circuit. The intention of this file is to prepare the expected results prior to the test and then compare the actual results to the calculated ".sim" file. This can be done during the display test feature of the DVS50. Once these files have been written the DVS50 menu driven program will be entered to test the chip.

D. TEST PROGRAM COMPILATION

The DVS50 is a menu driven interactive program that uses ".das" and ".src" files to generate test patterns and control the DAS 9100 tester. The command to enter the DVS50 program is: "dvs50 dvs50", illustrated in Figure 30 on page 75. The first menu that is encountered is the Main Menu and is illustrated in Figure 34 on page 81. To select a menu in this main menu move the cursor with the arrow keys and use < enter >. The Help menu will assist the new user in a brief summary of each menu item. The Update Configuration File controls the configuration, but the DVS50 is set up for the current configuration and this menu need not be entered.

The first menu usually entered is the Compile Test Program menu. After selection of this entry the program will prompt the user for the ".src" and ".das" files of the program to be run. The program will then compile these files and generate the DAS 9100 configuration and create a channel specification file. If the ".src" and ".das" files have been previously compiled and the test programs saved, then the test program can be restored by the Restore Test Program menu.

```

PROGRAM ALU181;
* test of the arithmetic logic unit 74xx181
PINDEF;
* pin description
B0      : 1,  PAT,  ACQ;
A0      : 2,  PAT,  ACQ;
S3      : 3,  PAT,  ACQ;
S2      : 4,  PAT,  ACQ;
S1      : 5,  PAT,  ACQ;
S0      : 6,  PAT,  ACQ;
CIN     : 7,  PAT,  ACQ;
M       : 8,  PAT,  ACQ;
F0      : 9,  ACQ;
F1      : 10, ACQ;
F2      : 11, ACQ;
F3      : 13, ACQ;
AEQB    : 14, ACQ;
P       : 15, ACQ;
COUT    : 16, ACQ;
G       : 17, ACQ;
B3      : 18, PAT,  ACQ;
A3      : 19, PAT,  ACQ;
B2      : 20, PAT,  ACQ;
A2      : 21, PAT,  ACQ;
B1      : 22, PAT,  ACQ;
A1      : 23, PAT,  ACQ;
GND     : 12, PS 1;
VDD     : 24, PS 2;
END;
TIMEDEF;
* Pattern Generator Clock Rate.
PAT : ns 200;
END;
THRESHOLD;
* Threshold for Acquisition Modules.
ACQ : TTL;
END;
PSDEF;
* Definition of Power Supply.
1   : mV 0;
2   : mV 5000, mA 3000;
END;
BEGIN;
END$

```

Figure 31. ALU181 ".src" File


```

ALU181
1
1
14
B3
B2
B1
B0
A3
A2
A1
A0
CIN
M
S3
S2
S1
S0
00000000101001
00010001101001
00100010101001
01000100101001
10001000101001
10010111101001
11001100101001
10011001101001
11111111101001

```

Figure 32. ALU181 ".das" File

E. DUT SET UP

Once the ".sim" and ".das" input test files have been compiled the information menu, Figure 34 on page 81, is entered to determine the channel specification definitions. It is recommended that the Printing item be selected first by pressing enter when the cursor is at that position. This will toggle the printing from disabled to enabled. Then select the channel specification entry, and the results will be printed. If nothing happens, check to see if the printer cable is connected to the PC that is the host to DVS50. The channel specification list gives the trigger, clock, pattern and data acquisition connections to the DUT. With the ALU181 channel specification, Figure 35 on page 83,

```

ALU181
1
1
19
B3
B2
B1
B0
A3
A2
A1
A0
CIN
M
S3
S2
S1
S0
COUT
F3
F2
F1
F0
0000000010100110000
0001000110100110010
0010001010100110100
0100010010100111000
1000100010100100000
1001011110100100000
1100110010100101000
1001100110100100010
1111111110100101110

```

Figure 33. ALU181 ".sim" File

the layout of the test jig, Figure 37 on page 85, and the module layout, Figure 38 on page 86, the user can connect the modules to the DUT.

Certain procedures need to be followed to ensure that damage is not done to the DUT or the DAS9100. Power will remain off during the entire set up procedure. The following steps are recommended for connecting the DUT, but they need not be accomplished in the order stated:

- *****ENSURE THAT THE POWER SUPPLY IS OFF*****
- Connect the grounds for the modules:

- two pomona hook clips for data acquisition
- two wires, black and green, from the pattern generator
- one pomona hook clip from the external clock
- Connect power for the pattern generator module, red wire to +5V
- Install the two 12 pin resistor pads and install them next to the 40 pin easy out, with the wire connected to pin 12, gnd, and pin 24, Vdd.
- Connect power and ground to the chip.
 - Connect a red wire from +5V to row D, pin 24 of the test jig.
 - Connect the capacitor between rows E and F of pin 24.
 - The ground is connected with a small black jumper from row A to B on pin 12.
- Hook up the trigger.
 - Pattern generator 1B STB-1, probe 8, is connected to Acquisition channel 5D7-1.
 - Use the jumper pin connections and ensure the reference side of the pattern generator connected to ground.
- Hook up External Clock.
 - Clk1 (black wire). 7C-CLK1, to pattern generator, 1B-CLK4 jumper pin connections
 - connect any ground to the reference side of the pattern generation probe.
- Install the ALU181 chip in the 40 pin connector. Ensure pins 1 and 24 are closest to the power supply connections.
- Connect the pattern generator probes to rows C and D of the appropriate pin
- Connect the data acquisition probes to A,B and E,F of the appropriate pins. Ensure that the reference line is to the outside of the test jig.

F. RUNNING THE TEST

Once the DUT is hooked up, turn the power on the DAS 9100 series tester. Then enter the Enter Test Menu illustrated in Figure 36 on page 84. Once in the Test menu select Run Test and then turn on the power supply when prompted by the program. The DVS50 program will download the channel specifications and the test pattern sequences to the DAS 9100. The DAS 9100 will then run the test on the DUT, ALU181, and then upload the acquired data to the DVS50 program. When this data transfer is complete, the DVS50 program will prompt the user to turn the power supply power off. This is necessary because the podlets, the small circuit boards at the end of the pattern generation probes, generate heat.

```

keys to move cursor: use <ENTER> to select.
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                                     Main Menu

```

```

sys to move cursor: use <ENTER> to select.
#####
                                Information Menu
                        Test Program in Use : ALU181
#####

```

81

Once the power supply is turned off, return to the Main menu and then enter the Display Test Menu illustrated in Figure 36 on page 84. The program prompts for the ".A01" and the ".sim" files. The ".A01" file contains the results of the test run by the DAS 9100. The ".sim" file is developed by the user that includes the expected circuit results. If no ".sim" file has been produced, then press the space bar and <enter> to indicate no additional file is to be compared.

The waveform in Figure 39 on page 87 is the display from the two 4-bit word addition test of the ALU181. This display can contain 24 pin names and corresponding waveforms. The display window can be expanded, compressed, or moved to a different time in the test sequence. Figure 40 on page 88 lists the additional commands available to modify this screen. The only way to print out the display is to invoke the print screen utility after the GRAFPLUS program has been installed. This installation must take place before entering the DVS50 Main menu. The commands for installing the GRAFPLUS utility are located in Figure 30 on page 75. Examination of the displayed test results should indicate that the ALU181 performed as designed.

G. CONCLUSIONS

The combination of the DAS 9100 and the DVS50 program offer the user a powerful digital analysis tool. A chip with up to 40 pins with TTL logic can be tested by generating test sequence patterns and acquiring data on 31 channels. The ALU181 demonstrated the testing methodology and verified the steps necessary to conduct a test. Further information about the DVS50 program and the DAS 9100 series tester can be obtained from the 91DVS User's Manual [Ref. 20] and the DAS 9100 Operator's Manual [Ref. 12].

ATTENTION : Connect the DAS-PODs to the pins of the DUT according to the following list.
Take care - incorrect connections may cause permanent damage to the DAS-PODs or the DUT !

```

*****
*
* Trigger channel(s):          PG-POD          ACQ-POD
*
*                             1B-STB-1        5D7-1
*
*****
* ACQ clock channel(s):       PG-POD          EXTCLK-POD
*
*                             1B-CLK-1        7C-CLK1-1
*
*****
*
* NR: NAME  PINNUMBER          POD(S)
*
*          PG  ACQN  ACQF      other
*  1  B0      1  1B7-1  5D6-1
*  2  A0      2  1B6-1  5D5-1
*  3  S3      3  1B5-1  5D4-1
*  4  S2      4  1B4-1  5D3-1
*  5  S1      5  1B3-1  5D2-1
*  6  S0      6  1B2-1  5D1-1
*  7  CIN     7  1B1-1  5D0-1
*  8  N       8  1B0-1  5C7-1
*  9  P       9         5C6-1
* 10  F1     10         5C5-1
* 11  F2     11         5C4-1
* 12  F3     13         5C3-1
* 13  AEQB   14         5C2-1
* 14  P      15         5C1-1
* 15  COUT   16         5C0-1
* 16  G      17         5B7-1
* 17  B3     18  1A7-1  5B6-1
* 18  A3     19  1A6-1  5B5-1
* 19  B2     20  1A5-1  5B4-1
* 20  A2     21  1A4-1  5B3-1
* 21  B1     22  1A3-1  5B2-1
* 22  A1     23  1A2-1  5B1-1
* 23  GND    12
* 24  VDD    24
*
*          PS1
*          PS2
*
*****

```

Figure 35. ALU181 Channel Specifications

[illegible]

Figure 36. DVS50 Test Menu

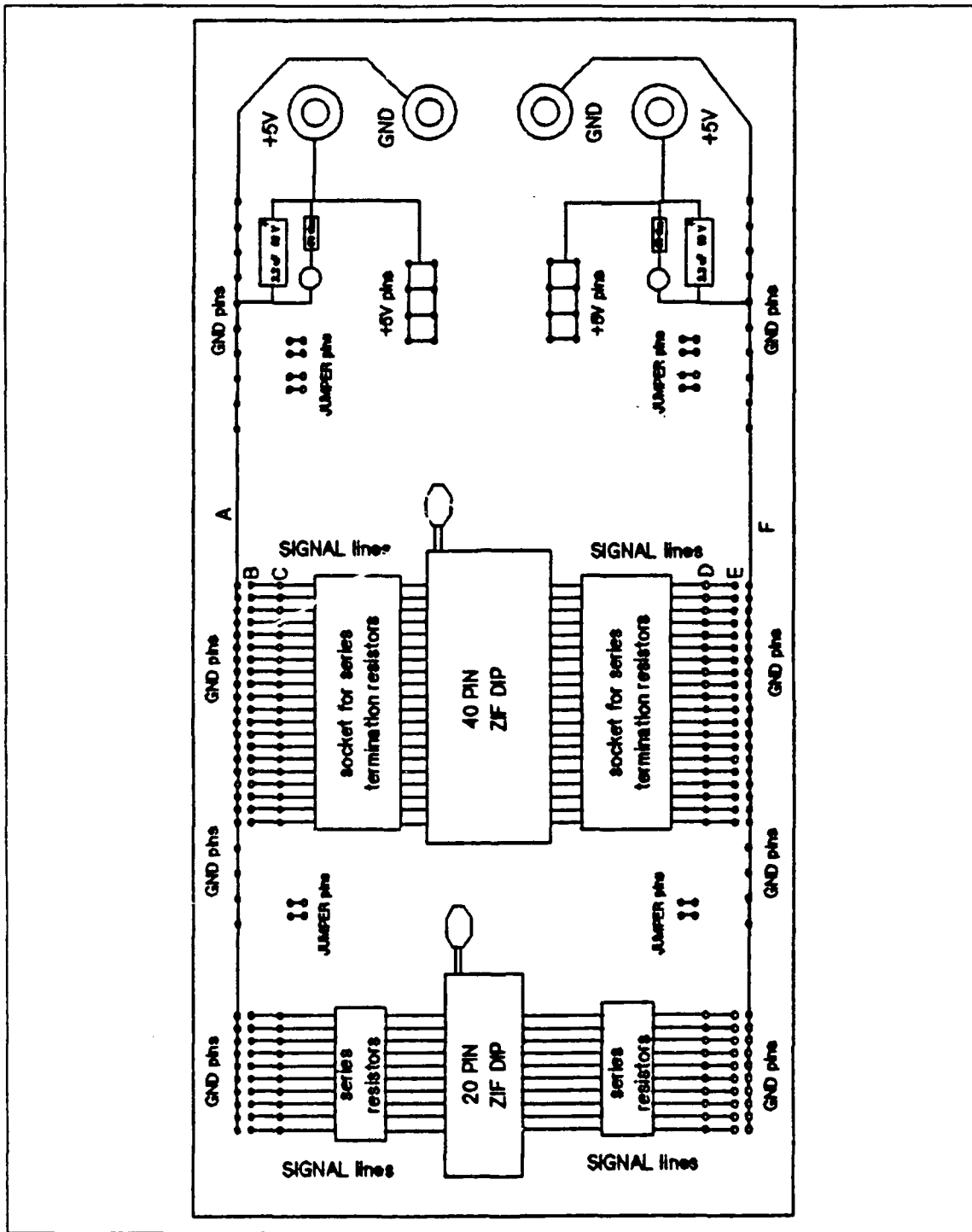


Figure 37. Test Jig

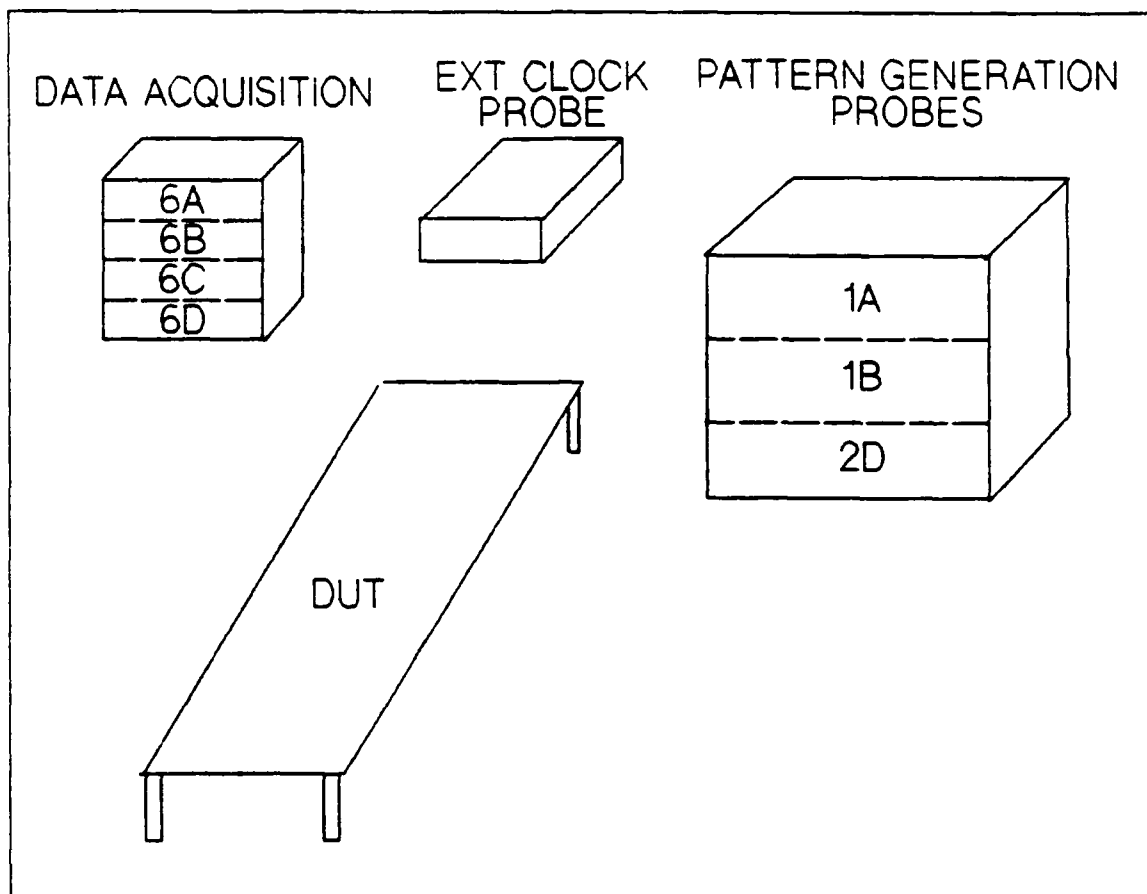


Figure 38. Physical Location of the Modules

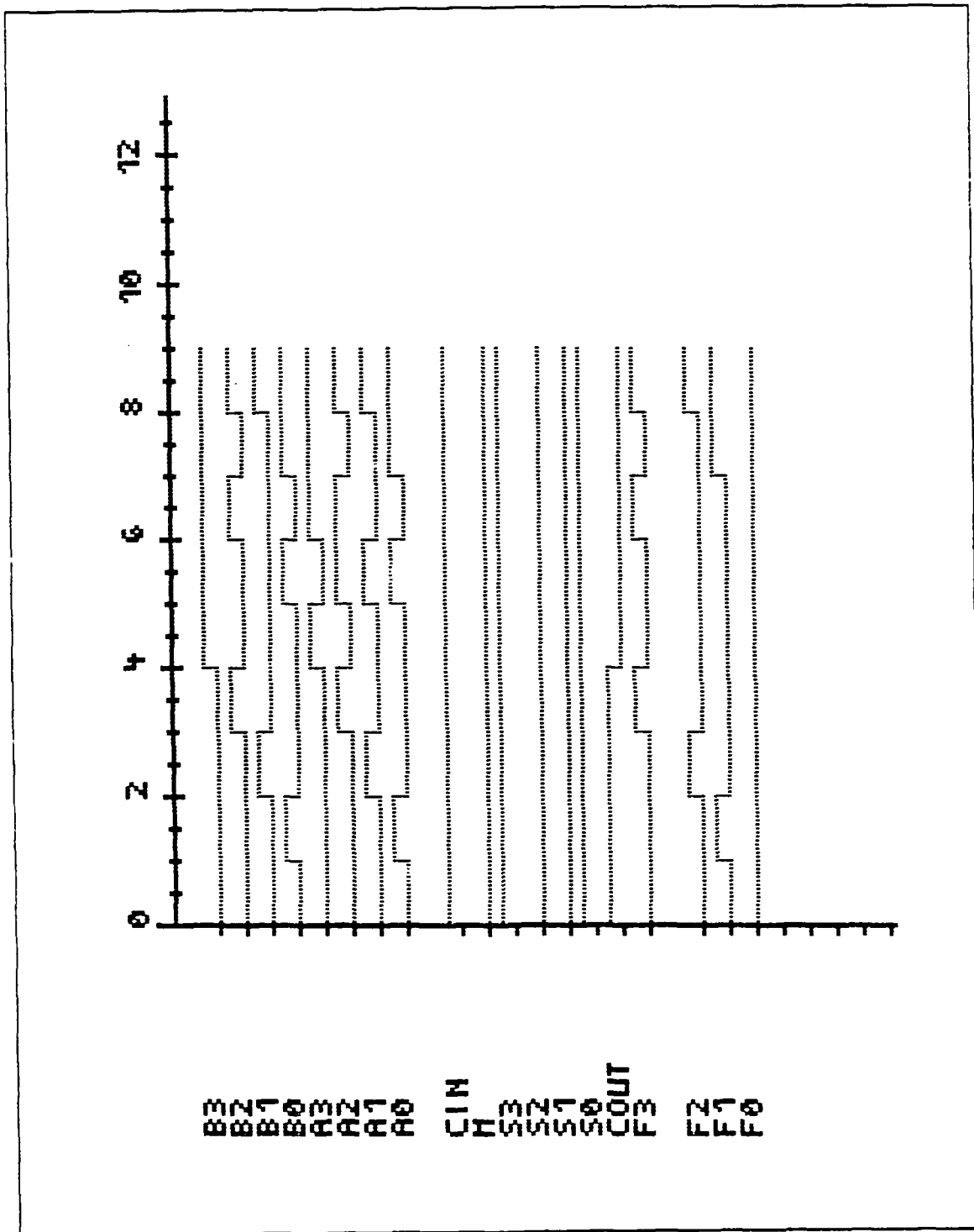


Figure 39. ALU181 Results

APPENDIX B. MOSTRANS TRANSLATION PROGRAM

This Appendix contains the "MOS2DAS" shell program that controls the file conversion process for the "mostral" Fortran program that translates the MOSSIM II program to DVS50 input programs. The "MOS2DAS" program requires the ".ntk" and ".cpy" files as input files and checks to see if they are valid files. The shell program then converts the ".ntk" file into an input and output pin list, NTKIN.DAT and NTKOUT.DAT. These files and the ".cpy" file renamed "CHECK" are used as input files for the "mostral" Fortran translator program.

The "mostral" program will read the "CHECK" program one line at a time and look for the key MOSSIM II commands. This program also generates an interactive session with the user to define the desired pins, pin numbers, power supply definitions, and timing requirements. The translator will then examine the "CHECK" file and converts it to the "MOS.DAS", "MOS.SRC", and "MOS.SIM" DVS50 input files.

The shell program then renames the translator files to the correct extensions to the ".ntk" file name. The final action of the shell program is to erase all working files.

A. TRANSLATOR SHELL PROGRAM

```
# this is the beginning of all C Shell programs
# this checks if there are two arguments
if( $#argv != 2)then
    echo "usage: ntk2das <file.ntk> <file.cpy>"
    exit
endif

set ntkfile=$1
#check to see if the input file is a valid ntk file
if(! -e $ntkfile ) then
    set ntkfile=$1.ntk
    if( ! -e $ntkfile)then
        echo "cannot find the file $1 or $1.ntk"
        exit
    endif
endif

#check to see if the input file is a valid mos copy file

set mosfile=$2
if(! -e $mosfile ) then
    set mosfile=$2.cpy
    if( ! -e $mosfile)then
```

```

        echo "cannot find the file $2 or $2.cpy"
        exit
    endif

endif

# convert the files to be used for the translator
egrep          i $ntkfile    awk '[print $2 ]' > NTKIN.DAT
egrep          s $ntkfile >pout1
egrep _ $ntkfile >pout2
diff pout2 pout1 > pout3
egrep '> s' pout3    awk '[print $4]' > NTKOUT.DAT
cp $mosfile CHECK
mostrat
cp MOS.DAS $ntkfile:r.das
cp MOS.SRC $ntkfile:r.src
cp MOS.CMP $ntkfile:r.sim
rm MOS.CMP MOS.SRC MOS.DAS CHECK
rm pout1 pout2 pout3 NTKIN.DAT NTKOUT.DAT

```

B. TRANSLATOR SOURCE PROGRAM

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC  PROGRAM      MOSTRANS                                     CC
CC  INPUTS:      NTKIN.DAT   INPUT PINS                      CC
CC                NTKOUT.DAT  OUTPUT PINS                   CC
CC                CHECK      MOSSIM II .CPY FILE             CC
CC  OUTPUTS:     MOS.DAS     CC
CC                MOS.SIM    CC
CC                MOS.SRC    CC
CC
CC  PURPOSE:      TRANSLATE THE MOSSIM II .CPY FILE, CHECK, AND CC
CC                TWO NTK FILES INTO THE THREE DVS50 FILES   CC
CC
CC  LIMITATIONS:
CC                MAX NUMBER OF DATA ACQUISITION CHANNELS   31 CC
CC                MAX NUMBER OF PATTERN GENERATION CHANNELS  47 CC
CC                MAX NUMBER OF REPETITIVE CYCLES             20 CC
CC                MAX NUMBER OF POWER SUPPLIES                9  CC
CC                MAX PIN NAME LENGTH                         10 CC
CC
CC  ***** TO CHANGE THE MAX NUMBER OF DATA ACQUISITION CHANNELS CC
CC                CHANGE MAXACQ IN SUBROUTINE SRC              CC
CC  ***** TO CHANGE THE MAX NUMBER OF PATTERN GENERATION CC
CC                CHANNELS CHANGE MAXPAT IN SUBROUTINE SRC     CC
CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

```

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC  ARRAY      DESCRIPTION                                     CC
CC
CC  NTKINPIN(40)  INPUT PIN NAMES FROM .NTK FILE             CC
CC  NTKOUTPIN(40) OUTPUT PIN NAMES FROM .NTK FILE             CC
CC  MOSINVEC(20)  INPUT VECTOR NAMES FROM MOSSIM FILE         CC
CC  MOSOUTVEC(20) OUTPUT VECTOR NAMES FROM MOSSIM FILE         CC
CC  VPOSIN(20)    POINTER TO ENTRY INTO MOSINVEC              CC
CC  VPOSTRT(20)   POINTER TO ENTRY INTO MOSOUTVEC             CC
CC  DASINPIN(40)  INPUT PIN NAMES TO .DAS                     CC
CC  DASOUTPIN(40) OUTPUT PIN NAMES TO .DAS                     CC
CC  CHECK(8)      HOLDS MOSSIM KEY WORDS                      CC
CC  ALINE(80)     HOLDS COMPLETE LINE FROM MOSSIM FILE        CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

```

```

INTEGER I,FSTCHAR, LASTCHAR, IVNUMIN, IVNUMOUT, IVPOSIN, IVPOSOUT
INTEGER VPOSIN(20), VPOSTRT(20), CLKSZ, FSTCYCLE, CYCLENM, FINCNT
INTEGER NTKIN, NTKOUT, IVWATCH, WPOSTRT(40)
INTEGER VPOSTOP(20), IWAT, WPOSTOP(40), SIMCNT, DASCNT, FSTDEL
INTEGER CLKCNT, DASINCNT
CHARACTER*80 SIMLINE, DASLINE, COMLINE, BLANK80
CHARACTER*24 WORD, BLANK24, TEMPW, CLOCKNM(10), CLOCKVA(10)

```

```

CHARACTER*24 CLKNUM,TCLKVAL
CHARACTER*40 DASIN,PDASIN,DASOUT
CHARACTER*10 INPUT,TENBLNK,WATPIN(80)
CHARACTER*1 OWATVAL(40),DWATVAL(40)
CHARACTER*10 MOSINVEC(20),MOSOUTVEC(20),VWATCH(40)
CHARACTER*10 NTKINPIN(40),NTKOUTPIN(40),DASINPIN(40),DASCUTPIN(40)
CHARACTER*10 SIMNM(40),DASNM(40)
CHARACTER*2 CYCVAL(20)
CHARACTER*1 VCOMLINE(80),WATVAL(40),DELAY
CHARACTER*1 ALINE(80),BLANK
DATA CYCVAL/'1','2','3','4','5','6','7','8','9','10','11','12',
*'13','14','15','16','17','18','19','20'/
DATA BLANK24/' '/
DATA BLANK/' '/
DATA BLANK80/' '/

```

```

OPEN(UNIT=10,ERR=950,STATUS='OLD',FILE='CHECK')
OPEN(UNIT=11,ERR=950,STATUS='OLD',FILE='NTKIN.DAT')
OPEN(UNIT=12,ERR=950,STATUS='OLD',FILE='NTKOUT.DAT')
OPEN(UNIT=13,ERR=951,STATUS='NEW',FILE='MOS.DAS')
OPEN(UNIT=14,ERR=951,STATUS='NEW',FILE='MOS.CMP')
OPEN(UNIT=15,ERR=951,STATUS='NEW',FILE='MOS.SRC')

```

```

WORD=BLANK24
INT=0
FSTCYCLE=0
PINLONG=0
NMDEL=0
DO 5 I=1,40
    DASINPIN(I)=TENBLNK
    NTKINPIN(I)=TENBLNK
    NTKOUTPIN(I)=TENBLNK

```

```

5    CONTINUE
    IVNUMIN=0
    IVPOSIN=0
    IVNUMOUT=0
    IVPOSOUT=0
    FSTDEL=0
    IWAT=0
C    OPEN AND READ THE NTK FILES THAT HAVE THE INPUT AND OUTPUT PINS
    READ(11,60,END=70)(NTKINPIN(I),I=1,40)
60   FORMAT(A10)
70   NTKIN=I
    CLOSE(11,ERR=950)
    READ(12,80,END=90)(NTKOUTPIN(I),I=1,40)
80   FORMAT(A10)
90   NTKOUT=I
    CLOSE(12,ERR=950)

100  FSTCHAR=0
    INT=INT+1
    LASTCHAR=0

C    READS IN A LINE FROM THE MOSFILE TO CONVERT

```

```

      READ(10,110,END=950)ALINE
110  FORMAT(80A1)
CC   LOOKS FOR KEY MOSSIM II COMMANDS
      CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
      IF(WORD.EQ. '>VECTOR'.OR. WORD.EQ. '>vector')GOTO 400
      IF(WORD.EQ. '>FORCE'.OR. WORD.EQ. '>force')GOTO 600
      IF(WORD.EQ. '>WATCH'.OR. WORD.EQ. '>watch')GOTO 500
      IF(WORD.EQ. '>CLOCK'.OR. WORD.EQ. '>clock')GOTO 700
      IF(WORD.EQ. '>CYCLE'.OR. WORD.EQ. '>cycle')GOTO 750
      IF(WORD.EQ. '>QUIT' .OR. WORD.EQ. '>quit')GOTO 999
      GOTO 100

CCC  VECTOR HANDLING AREA
400  TEMPW=BLANK24
      CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
      TEMPW=WORD
C    CALL TO GET NEXT PIN NAME
      CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
C    CHECK TO SEE IF PIN WAS INPUT OR OUTPUT PIN
C    AND THAT WILL DETERMINE WHICH VECTOR TO FILL

      DO 410 I=1,40
          IF(WORD .EQ. NTKINPIN(I))GO TO 420
          IF(WORD .EQ. NTKOUTPIN(I))GO TO 460
410  CONTINUE
      PRINT *,'ERROR IN PIN FILE'
      GOTO 950

      420  IVNUMIN=IVNUMIN+1
          MOSINVEC(IVNUMIN)=TEMPW
          IVPOSIN=IVPOSIN+1
          VPOSIN(IVNUMIN)=IVPOSIN
      425  DASINPIN(IVPOSIN)=WORD
          CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
          IF(WORD .NE. 'NOWORD')THEN
              IVPOSIN=IVPOSIN+1
              GOTO 425
          ENDIF
          GOTO 100
      460  IVNUMOUT=IVNUMOUT+1
          MOSOUTVEC(IVNUMOUT)=TEMPW
          IVPOSOUT=IVPOSOUT+1
          VPOSTRT(IVNUMOUT)=IVPOSOUT
      465  DASOUTPIN(IVPOSOUT)=WORD
          CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
          IF(WORD .NE. 'NOWORD')THEN
              IVPOSOUT=IVPOSOUT+1
              GOTO 465
          ENDIF
          VPOSTOP(IVNUMOUT)=IVPOSOUT
          GOTO 100

CC   THIS AREA IS THE WATCH SECTION
500  CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
      IF(WORD .EQ. 'NOWORD')GOTO 100

```



```

DO 505 I=1,IVNUMIN
505 IF(WORD .EQ. MOSINVEC(I))GOTO 500
DO 510 J=1,NTKIN
510 IF(WORD .EQ. NTKINPIN(J))GOTO 500
DO 515 I=1,IVNUMOUT
    IF(WORD .EQ. MOSOUTVEC(I))THEN
        IVWATCH=IVWATCH+1
        IWAT=IWAT+1
        VWATCH(IVWATCH)=WORD
        WPOSTRT(IVWATCH)=IWAT
        DO 514 J=VPOSTRT(I),VPOSTOP(I)
            WATPIN(IWAT)=DASOUTPIN(J)
            IWAT=IWAT+1
514        CONTINUE
            IWAT=IWAT-1
            WPOSTOP(IVWATCH)=IWAT
            GOTO 500
        ENDIF
515 CONTINUE

C    THIS WILL CHECK IF THE OUTPIN HAS BEEN USED IN A VECTOR TO PREVENT
C    DUPLICATION OF PINS

DO 516 I=1,IVPOSOUT
516 IF(WORD.EQ.DASOUTPIN(I))GOTO 500
CONTINUE

C    THIS AREA WILL CHECK TO SEE IF YOU WANT TO WATCH AN OUTPUT PIN
C    NOT IN A VECTOR

DO 520 J=1 NTKOUT
    IF(WORD .EQ. NTKOUTPIN(J))THEN
        IWAT=IWAT+1
        WATPIN(IWAT)=WORD
        GOTO 500
    ENDIF
520 CONTINUE
GOTO 500

CC    THIS AREA IS THE FORCE ACTION AREA
CC    TWO THINGS CAN HAPPEN WHEN A FORCE COMMAND IS FOUND
CC    1. THE FORCE IS ON A VECTOR OR
CC    2. THE FORCE IS ON A SINGLE PIN
CC        A. HAS THE PIN BEEN IDENTIFIED BEFORE
CC        B. NEW PIN TO BE FORCED

600 CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
CC    NXTWORD IS EITHER A VECTOR NAME OR A INPUT PIN NAME
CC    CHECKING FOR VECTOR NAME
DO 620 I=1,IVNUMIN-1
    IF(WORD .EQ. MOSINVEC(I))THEN
        CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
        DASIN(VPOSIN(I):(VPOSIN(I+1)-1))=WORD(1:LASTCHAR-FSTCHAR+1)
        GOTO 100
    ENDIF
620 CONTINUE

```

```

        IF(WORD .EQ. MOSINVEC(IVNUMIN))THEN
            CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
            DASIN(VPOSIN(IVNUMIN):VPOSIN(IVNUMIN)+LASTCHAR-FSTCHAR)=
*           WORD(1:LASTCHAR-FSTCHAR+1)
            GOTO 100
        ENDIF

CC      FORCE A PIN THAT IS NOT ASSOCIATED WITH A VECTOR
CC      CHECK TO SEE IF THAT PIN HAD BEEN IDENTIFIED IN A VECTOR STATEMENT
DO 630 I=1,IVPOSIN
        IF(DASINPIN(I) .EQ. WORD)THEN
            CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
            DASIN(I:I) = WORD(1:1)
            GOTO 100
        ENDIF
630     CONTINUE

CC      NEW INPUT PIN IDENTIFIED PUT INTO ARRAY AND GET ASSOCIATED VALUE
IVPOSIN=IVPOSIN+1
DASINPIN(IVPOSIN)=WORD
CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
DASIN(IVPOSIN:IVPOSIN)=WORD(1:1)
GOTO 100

CC      ***** CLOCK *****
CC      CLOCK AREA WILL ESTABLISH VALUES FOR CLOCK AND CLOCKSIZ
700     CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
        IF(WORD .EQ. 'NOWORD')GOTO 100
        CLKCNT=CLKCNT+1
        CLOCKNM(CLKCNT)=WORD
        CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
        CLOCKVA(CLKCNT)=WORD
        CLKSZ=LASTCHAR-FSTCHAR+1
        GOTO 700

CC      ***** CYCLE *****
CC      CYCLE AREA WILL INSERT CLOCK AND OUTPUT DAS LINE
750     CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
        CLKNUM=WORD
        FINCNT=IVPOSIN
        DASINCNT=FINCNT+CLKCNT
        PDASIN=DASIN
753     DO 1753 JJ=1,CLKCNT
        DASINPIN(FINCNT+JJ)=CLOCKNM(JJ)
1753     CONTINUE

C      THE FIRST TIME CYCLE IS CALLED INPUT PINS ARE DUMPED TO MOS.DAS
C      AND THE COMBINATION OF INPUT AND WATCH PINS ARE DUMPED TO MOS.CMP

        IF(FSTCYCLE .EQ. 0)THEN
            CALL SRC(WATPIN,IWAT,DASINPIN,DASINCNT,SIMNM,SIMCNT,
*           DASNM,DASCNT,DELAY)
            WRITE(13,754)(DASNM(I),I=1,DASCNT)
754     FORMAT(A10)

```

```

        WRITE(14,757)(SIMNM(I),I=1,SIMCNT)
757      FORMAT(A10)
        FSTCYCLE=1
      ENDIF
      DO 760 I=1,20
760      IF(CLKNUM.EQ.CYCVAL(I))CYCLENM=I
      DO 780 J=1,CYCLENM

        DO 770 III=1,CLKSZ
          FSTCHAR=0
          LASTCHAR=0
        C      GOTO FILL AREA TO FILL NEXT LINE WITH SIM OUT DATA
900      READ(10,901,END=950)ALINE
901      FORMAT(80A1)
902      CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
        C      LOOK FOR OUTPUT WATCH VECTOR TO MATCH
          IF(WORD.EQ.'NOWORD')GOTO 763
          DO 930 I=1,IWATCH
            IF(WORD.EQ.VWATCH(I))THEN
              CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
              K=1
              DO 920 JJ=WPOSTRT(I),WPOSTOP(I)
                WATVAL(JJ)=WORD(K:K)
                K=K+1
920              CONTINUE
              GOTO 902
            ENDIF
930          CONTINUE

        C      OUTPUT FPIN FOUND IN SIMULATOR OUTPUT LINE
          DO 940 JJ=1,IWAT
            IF(WORD.EQ.WATPIN(JJ))THEN
              CALL NXTWORD(ALINE,FSTCHAR,LASTCHAR,WORD)
              WATVAL(JJ)=WORD(1:1)
              GOTO 902
            ENDIF
940          CONTINUE

          GOTO 902
763      DO 1763 JJ=1,CLKCNT
          TCLKVAL=CLOCKVA(JJ)
          PDASIN(FINCNT+JJ:FINCNT+JJ)=TCLKVAL(III:III)
1763      CONTINUE
          SIMLINE=BLANK80
          DASLINE=BLANK80
        C      COMBINE THE INPUT PINS AND WATCH PINS FOR COMPARE FILE
          DO 766 IL=1,DASCNT
            DO 765 JI=1,DASINCNT
              IF(DASNM(IL).EQ.DASINPIN(JI))DASLINE(IL:IL)=PDASIN(JI:JI)
765            CONTINUE
766          CONTINUE
          DO 1767, LL=1,DASCNT
1767          IF(DASLINE(LL:LL).NE.'1')DASLINE(LL:LL)='0'
          WRITE(13,767)DASLINE
767          FORMAT(A80)
          DO 769 IL=1,SIMCNT

```

```

DO 768 JI=1,IWAT
IF(DELAY .EQ. 'Y' .OR. DELAY .EQ. 'y')THEN
    IF(FSTDEL .EQ. 0)THEN
        OWATVAL(JI)=WATVAL(JI)
    ELSE
        OWATVAL(JI)=DWATVAL(JI)
    ENDIF
    DWATVAL(JI)=WATVAL(JI)
ELSE
    OWATVAL(JI)=WATVAL(JI)
ENDIF
IF(SIMNM(IL) .EQ. WATPIN(JI))THEN
    SIMLINE(IL: IL)=OWATVAL(JI)
    GOTO 1769
ENDIF
768    CONTINUE
1769    DO 772 ILL=1,DASINCNT
        IF(SIMNM(ILL) .EQ. DASINPIN(ILL))SIMLINE(IL: ILL)=
* PDASIN(ILL: ILL)
772    CONTINUE
769    CONTINUE
        FSTDEL=1
        DO 1171, LL=1,SIMCNT
1171    IF(SIMLINE(LL: LL) .NE. '1')SIMLINE(LL: LL)='0'
        WRITE(14,771)SIMLINE
771    FORMAT(A80)
770    CONTINUE
780    CONTINUE
        GOTO 100

CC    THIS AREA IS WHERE THE OUTPUT VALUES ARE STORED IN THE OUTPUT
CC    VARIABLES OF DASOUTPIN

950    PRINT *, 'INPUT ERROR PROCESSED'
        GOTO 999
951    PRINT *, 'NEWFILE ERROR'

        CLOSE(13,ERR=999)
        CLOSE(14,ERR=999)
        CLOSE(15,ERR=999)
999    STOP
        END

```

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC      PROGRAM:      MOSTRANS      CC
CC      SUBROUTINE:  NXTWORD      CC
CC      INPUTS:      ALINE      CHARACTER LINE OF DATA      CC
CC      FSTCHAR      FIRST CHARACTER POSITION OF LOOK AT      CC
CC      LASTCHAR      LAST CHARACTER POSITION PRIOR TO      CC
CC      DELINEATOR      CC
CC      OUTPUT:      WORD      WORD OF INTEREST      CC
CC      PURPOSE:      TO LOOK AT A CHARACTER STRING AND USING THE SPACE      CC
CC      AND COLON DELINEATORS PARSE THE NEXT WORD      CC
CC      CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

```

```

      SUBROUTINE NXTWORD(ALINE,FSTCHAR, LASTCHAR,WORD)
      INTEGER FSTCHAR, LASTCHAR, ZEROCH
      CHARACTER*1 ALINE(80), BLANK, COLON
      CHARACTER*24 WORD, BLANK24
      DATA BLANK/' ', BLANK24/'
      DATA COLON/':',
      FSTCHAR=LASTCHAR+1
      ZEROCH=0
      WORD=BLANK24
      DO 200 I=FSTCHAR,80
         IF(ALINE(I) .EQ. BLANK .OR. ALINE(I) .EQ. COLON)THEN
            GOTO 200
         ELSE
            IF(ZEROCH .EQ. 0)FSTCHAR=I
            ZEROCH=1
         ENDIF
200    CONTINUE
      IF(ZEROCH .EQ. 0)THEN
         WORD='NOWORD'
         GOTO 240
      ENDIF

      DO 210 J=FSTCHAR,80
         IF(ALINE(J) .EQ. BLANK .OR. ALINE(J) .EQ. COLON)THEN
            LASTCHAR=J-1
            GOTO 220
         ENDIF
210    CONTINUE
220    II=1
      DO 230 K=FSTCHAR, LASTCHAR
         WORD(II: II)=ALINE(K)
         II=II+1
230    CONTINUE
240    END

```

```

CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
CC      PROGRAM:      MOSTRANS      CC
CC      SUBROUTINE:    SRC      CC
CC      INPUTS:        WATPIN      ARRAY OF WATCH PINS      CC
CC                      IWAT      NUMBER OF WATCH PINS      CC
CC                      DASINPIN   ARRAY OF INPUT PINS      CC
CC                      DASINCNT   NUMBER OF INPUT PINS      CC
CC      OUTPUTS:       SIMNM      ARRAY OF OUTPUT PINS      CC
CC                      SIMCNT     NUMBER OF OUTPUT PINS      CC
CC                      DASNM      ARRAY OF PATTERN PINS      CC
CC                      DASCNT     NUMBER OF PATTERN PINS      CC
CC                      DELAY      ONE PHASE DELAY      CC
CC
CC      PURPOSE:       INTERACTIVE PROGRAM THAT WILL LIST THE INPUT AND      CC
CC                      WATCH PINS AND THE USER WILL PICK THE ONES THAT      CC
CC                      SHOULD BE TRANSLATED.  A DELAY CAN ALIGN THE ".SIM" CC
CC                      AND ".A01" DVS50 FILES.      CC
CC
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC

```

```

      SUBROUTINE SRC(WATPIN,IWAT,DASINPIN,DASINCNT,SIMNM,SIMCNT,
*      DASNM,DASCNT,DELAY)
      INTEGER I,IWAT,SRCPIN(40),DASINCNT,SIMCNT,SRCCNT,DASCNT
      INTEGER PSCNT,PSAMP(10),PSVOLT(10),TIMEVAL,MAXPAT,MAXACQ
      CHARACTER*10 WATPIN(5),SIMNM(40),DASINPIN(5),SRCNM(40)
      CHARACTER*10 DASNM(40),TENBLNK,PSNAME,CHIPNM
      CHARACTER*3 ATTA(40)
      CHARACTER*4 ATTB(40)
      CHARACTER*1 APUN(40),PATMON,ACQMON,DELAY
      DATA TENBLNK/'          '/

      MAXPAT=47
      MAXACQ=31
      DASCNT=0
      SRCCNT=0
      SIMCNT=0
      DO 10, I=1,40
      DASNM(I)=TENBLNK
      SIMNM(I)=TENBLNK
      SRCNM(I)=TENBLNK
      ATTA(I)=' '
      ATTB(I)=' '
      APUN(I)=' '
10    CONTINUE
CC    THIS STARTS THE DATA ACQUISITION SECTION
CC    THE USER WILL BE GIVEN THE MOSSIM II WATCH PINS
CC    AND HE WILL DECIDE WHETHER TO USE THE PIN AND GIVE THE
CC    PIN NUMBER
CC          SRCNM ARRAY      HOLDS PINS FOR THE .SRC FILE
CC          SIMNM ARRAY      HOLDS PINS FOR THE .SIM FILE
CC          SRCCNT,SIMCNT    COUNTS NUMBER OF PINS IN EACH ARRAY

      WRITE(6,201)
201  FORMAT(1X,'WHAT IS THE NAME OF YOUR FILE?',/,
*1X,'EX. CHIP ***NO EXTENSION REQUIRED ')

```

```

      READ(5,208)CHIPNM
208  FORMAT(A10)
      WRITE(15,209)CHIPNM
209  FORMAT('PROGRAM',1X,A10,/, '* ',/, '* ',/, 'PINDEF;',/, '* ')
      WRITE(6,11)
11   FORMAT(1X,'THERE ARE A MAXIMUM OF 31 DATA ACQUISITION, (MONITOR), '
*,/, ' AND 31 PATTERN GENERATION CHANNELS FOR THIS DAS 9100 AND '
*,/, ' DVS50 SOFTWARE CONFIGURATION')
      DO 50, I=1,IWAT
          WRITE(6,15)WATPIN(I)
15      FORMAT(1X,'DO YOU WANT TO MONITOR ',A10,'?')
          READ(5,18)ACQMON
18      FORMAT(A1)
          IF(ACQMON .EQ. 'N' .OR. ACQMON .EQ. 'n')GOTO 50
          SRCCNT=SRCCNT+1
          SIMCNT=SIMCNT+1
          IF(SIMCNT .GE. 32)GOTO 401
          SRCNM(SRCCNT)=WATPIN(I)
          SIMNM(SIMCNT)=WATPIN(I)
          ATTA(SRCCNT)='ACQ'
          APUN(SRCCNT)=';'
          WRITE(6,20)
20      FORMAT(1X,'WHAT IS THE PIN NUMBER? ')
          READ(5,30)SRCPIN(SRCCNT)
30      FORMAT(I2)
50      CONTINUE

CC     THIS AREA WILL INPUT THE PATTERN PINS
CC     AND WILL ASK THE USER IF THE PIN IS TO BE INCLUDED
CC     AND IF MONITORING IS DESIRED

      DO 100, I=1,DASINCNT
          WRITE(6,60)DASINPIN(I)
60      FORMAT(1X,'DO YOU WANT A PATTERN ON PIN ',A10,'?')
          READ(5,65)PATMON
65      FORMAT(A1)
          IF(PATMON .EQ. 'N' .OR. PATMON .EQ. 'n')GOTO 100
          SRCCNT=SRCCNT+1
          DASCNT=DASCNT+1
          IF(DASCNT .GE. 32)GOTO 403
          SRCNM(SRCCNT)=DASINPIN(I)
          DASNM(DASCNT)=DASINPIN(I)
          ATTA(SRCCNT)='PAT'
          WRITE(6,70)
70      FORMAT(1X,'WHAT IS THE PIN NUMBER? ')
          READ(5,75)SRCPIN(SRCCNT)
75      FORMAT(I2)
          WRITE(6,80)
80      FORMAT(1X,'DO YOU ALSO WANT TO MONITOR THIS PIN? ')
          READ(5,85)ACQMON
85      FORMAT(A1)
          IF(ACQMON .EQ. 'N' .OR. ACQMON .EQ. 'n')THEN
              APUN(SRCCNT)=';'
              GOTO 100
          ELSE
              SIMCNT=SIMCNT+1

```

```

                IF(SIMCNT .GE. 32)GOTO 401
                SIMNM(SIMCNT)=DASINPIN(I)
                APUN(SRCCNT)=' '
                ATTB(SRCCNT)='ACQ; '
            ENDIF

100    CONTINUE

CC     THIS IS THE POWER SUPPLY SECTION
        PSCNT=0
105    WRITE(6,110)
110    FORMAT(1X,'THE USER WILL DEFINE THE POWER SUPPLY PARAMETERS',/,1X,
        *' IN THIS SECTION. WHAT IS A PIN NAME FOR THE POWER SUPPLY?',/,
        *' EX. VDD OR GND ***TYPE DONE FOR PIN NAME WHEN FINISHED ',/)

        READ(5,115)PSNAME
115    FORMAT(A10)
        IF(PSNAME .EQ. 'DONE' .OR. PSNAME .EQ. 'done')GOTO 200
        PSCNT=PSCNT+1
        SRCCNT=SRCCNT+1
        SRCNM(SRCCNT)=PSNAME
        WRITE(6,70)
        READ(5,120)SRCPIN(SRCCNT)
120    FORMAT(I2)
        ATTA(SRCCNT)='PS '
        IF(PSCNT .EQ. 1)APUN(SRCCNT)='1'
        IF(PSCNT .EQ. 2)APUN(SRCCNT)='2'
        IF(PSCNT .EQ. 3)APUN(SRCCNT)='3'
        IF(PSCNT .EQ. 4)APUN(SRCCNT)='4'
        IF(PSCNT .EQ. 5)APUN(SRCCNT)='5'
        IF(PSCNT .EQ. 6)APUN(SRCCNT)='6'
        IF(PSCNT .EQ. 7)APUN(SRCCNT)='7'
        IF(PSCNT .EQ. 8)APUN(SRCCNT)='8'
        IF(PSCNT .EQ. 9)APUN(SRCCNT)='9'
        ATTB(SRCCNT)='; '
        WRITE(6,125)
125    FORMAT(1X,'WHAT VOLTAGE, IN mV, IS ON THIS PIN?',/,1X,
        *'EX 5000 ***NOTE GND MUST HAVE 0 ')
        READ(5,130)PSVOLT(PSCNT)
130    FORMAT(I5)
        WRITE(6,135)
135    FORMAT(1X,'WHAT AMPERAGE, IN mA, IS ON THIS PIN?',/,1X,
        *'EX 3000 ***NOTE GND MUST HAVE 0 ')
        READ(5,140)PSAMP(PSCNT)
140    FORMAT(I5)
        GOTO 105
200    WRITE(15,210)(SRCNM(I),SRCPIN(I),ATTA(I),APUN(I),
        *ATTB(I),I=1,SRCCNT)
210    FORMAT(A10,': ',I2,', ',2x,A3,A1,A4)
CC     THIS SECTION WILL INPUT THE TIMING REQUIREMENTS
        WRITE(15,220)
220    FORMAT('END; ',/, '* ',/, 'TIMEDEF; ',/, '* ',/
        *, '* CLOCKRATE PATTERN GENERATOR. ',/, '* ')
        WRITE(6,225)
225    FORMAT(1X,'WHAT CLOCKING RATE, IN nS, DO YOU WANT FOR THE'
        *, ' CIRCUIT?',/, ' FROM 40nS TO 5000nS EX. 100 ')

```



```

      READ(5,230)TIMEVAL
230  FORMAT(I4)
      WRITE(13,231)CHIPNM,TIMEVAL,TIMEVAL,DASCNT
231  FORMAT(A10,/,I4,/,I4,/,I3)
      WRITE(14,232)CHIPNM,TIMEVAL,TIMEVAL,SIMCNT
232  FORMAT(A10,/,I4,/,I4,/,I3)
      WRITE(15,235)TIMEVAL
235  FORMAT('PAT : ns ',I4,',')
      WRITE(15,240)
240  FORMAT('END; ',/, '* ',/, 'THRESHOLD; ',/, '* ',/,
  '* THRESHOLD FOR ACQUISITION MODULE. ',/, '* ',/, 'ACQ : TTL; ',/,
  '* END; ')
      WRITE(15,202)
202  FORMAT('* ',/, 'PSDEF ',/, '* ',/, '* DEFINITION OF THE'
  *, 'POWER SUPPLY. ',/)
      WRITE(15,205)(I,PSVOLT(I),PSAMP(I),I=1,PSCNT)
205  FORMAT(I1, ' : mV ',I5,', mA ',I5,',')
      WRITE(15,206)
206  FORMAT('END; ',/, '* ',/,
  '* ',/, '* ',/, 'BEGIN; ',/, 'END$')
      WRITE(6,410)
410  FORMAT(1X,'DO YOU WANT A ONE PHASE DELAY IN THE OUTPUT TO'
  *, ' ALIGN THE OUTPUT FILES FOR MOSSIM AND DAS 9100? 'Y,N ' )
      READ(5,415)DELAY
415  FORMAT(A1)
      GOTO 500
401  WRITE(6,402)
402  FORMAT(1X,' *****ERROR***** YOU HAVE EXCEEDED THE NUMBER',
  *, ' OF PINS TO MONITOR',/, ' YOU MUST RUN THE TRANSLATOR AGAIN. ')
      GOTO 500
403  WRITE(6,404)
404  FORMAT(1X,' *****ERROR***** YOU HAVE EXCEEDED THE NUMBER',
  *, ' OF PINS FOR PATTERN',/, ' GENERATION. YOU MUST RUN THE',
  *, ' TRANSLATOR AGAIN. ')

500  END

```

APPENDIX C. VECTOR GENERATION PROGRAMS

A. VECTOR SHELL PROGRAM

```
#
#This program will take vectors in vectors.out
#and input them to the MOSSIM II program crm
#and then compare the results the MOSSIM II
#program with a comvec.out and put the differences in
#a file called difres.out
#
echo ""
echo "start time of the program was ">difres.out
date>>difres.out
echo ""
rm -f mossim.nst mossim.bst
echo ""
echo "Disregard the below comments from Mossim."
echo "Be patient."
echo ""
mossim << +
sour crm
yes
sour vectors.out
q
+
edit CR_result.cpy<resmod.cmd>/dev/null
echo ""
echo "Only differences in the files are listed below.3" >> difres.out
echo -n "If there are no differences your files are matched." >> difres.out

echo ""
diff -b CR_result.cpy comvec.out >> difres.out
echo "end time is ">>difres.out
date>>difres.out
else
echo ""
exit
echo ""
endif
```

B. VECTOR GENERATION C SOURCE CODE

```
/*This program will generate the number of 16-bit vectors that
is requested by the user. These vectors are written to the
vectors.out file. This program also computes the values for
the output vector from the 16-bit correlator, NPS CORN88.
These output vectors are located in the comvec.out file. */

#include<stdio.h>
#include<math.h>

float i;
float t2;
int j,k,l;
unsigned int mask=1,vec,nout;
int temp[16],cout[5];
int t1,rtime;

main()
{
    FILE *fp1, *fopen();
    FILE *fp2, *fopen();
    fp1=fopen("vectors.out","w");
    fp2=fopen("comvec.out","w");

    printf("How large do you want the vector. n");
    scanf("%6d",&t1);
    /*
    sscanf(argv[1],"%5d",&t1);*/
    fprintf(fp1,"force control:1000 n");
    fprintf(fp1,"force select:000 n");
    fprintf(fp1,"force misc:11 n");

    for(i=0;i<t1;i++){
        vec=i;
        for(j=0;j<=15;j++){
            if(vec&mask)temp[j]=1;
            else temp[j]=0;
            vec=vec>>1;
        }
        fprintf(fp1,"force ins:");
        for(k=15;k>=1;k--){
            fprintf(fp1,"%d",temp[k]);
        }
        fprintf(fp1,"%d n",temp[0]);
        fprintf(fp1,"cycle n");
    }

    /* computes the compare vectors and outputs to comvec.out*/
    rtime=20;
    for(i=0;i<t1;i++){
        vec=i;
        rtime=rtime+1;
        nout=0;
```

```

        for( j=0; j<=15; j++){
if(vec&mask){temp[ j]=1;
        nout=nout+1;}
        else temp[ j]=0;
        vec=vec>>1;
    }

    for( l=0; l<=4; l++){
        if(nout&mask)cout[ l]=1;
        else cout[ l]=0;
        nout=nout>>1;
    }

        if(rtime<=99)
fprintf(fp2," %d",rtime);
        else
fprintf(fp2,"%d",rtime);
fprintf(fp2,".3  outs:");
for( l=4; l>=0; l--){
    fprintf(fp2,"%d",cout[ l] );
}
fprintf(fp2," ins:");
for(k=15; k>=1; k--){
    fprintf(fp2,"%d",temp[ k] );
}
fprintf(fp2,"%d n",temp[ 0] );
}

rewind(fp1);
rewind(fp2);
fclose(fp1);
fclose(fp2);

```

```

}

```

LIST OF REFERENCES

1. Reghbati, Hassan K., *Tutorial: VLSI Testing & Validation Techniques*, IEEE Computer Society, 1985
2. Designers' Buying Guide *Computer Design*, pp. 101-109, June 1, 1988
3. Scott, Walter S.; Mayo, Robert N.; Hamachi, Gordon; and Ousterhout, John K., *1986 VLSI TOOLS: Still More Works by the Original Artists*, Report No. UCB.CSD 86 272, Computer Science Division, Electrical Engineering and Computer Sciences, University of California Berkeley, California, December 1985
4. *Genesil System, System Description User's Manual*, Silicon Compiler Systems Corporation, San Jose California, September 1987
5. Settle, R.H., *Design Methodology Using the Genesil Silicon Compiler*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1988
6. Rockey, R.R., *Silicon Compiler Implementation of a Kalman Filter Algorithm as an ASIC*, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1988
7. Davidson, John, *Implementation of Design for Testability Strategy Using the Genesil Silicon Compiler*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1989
8. Bryant, Randy; Schuster, Mike; and Whitting, Doug, *MOSSIM II: A Switch-Level Simulator for MOS LSI User's Manual*, Massachusetts Institute of Technology, May 12, 1986
9. *RNL 4.2 (UW) User's Guide*, Northwest LIS Release 3.1, February 15, 1987

10. Vladimirescu, A.; Zhang K.; Newton, A.R.; Pederson, D.O.; Sangiovanni-Vincentalli, A., *Spice User's Guide*, Department of Electrical Engineering and Computer Science, University of California, Berkeley, California, February 15, 1987
11. *MOSIS user Manual, Release 3.0*, University of Southern California, Los Angeles, California, 1988
12. *DAS 9100 series, Operators Manual with Options*, Manual # 070-3624-01, TEKTRONIX, INC., Beaverton, Oregon, August 1986
13. Beck, Terence A., and Galinas, William J., "16-Bit VLSI Correlator Chip Design", Unpublished Technical Report, Naval Postgraduate School, March 1988
14. Scott, Walter S.; Mayo, Robert N.; Hamachi, Gordon; Ousterhout, John K.; Taylor, George S., *The Magic VLSI Layout System*, IEEE Design & Test, The Institute of Electrical Engineers, Inc., February 1985
15. Weste, Neil; Eshraghian, Kamran, *Principles of CMOS VLSI Design A Systems Perspective*, Addison-Wesley Publishing Company, October, 1985
16. Coordinate Free Lap Reference Manual, Version 1-2, Northwest LIS Release 3.1, February 15, 1987
17. Tomovich, Christine, *IEEE Circuits and Devices Magazine*, "MOSIS--A Gateway to Silicon", VOL. 4, NO. 2, pp. 22-23, March 1988
18. Carleton, David A, *A Pad Frame Generator with Automatic Channel Routing for VLSI Chip Assembly*, Master's Thesis, Naval Postgraduate School, Monterey California, December 1988
19. Hnatek, Eugene R.; Wilson, Beau R., *VLSI Design*, "Practical Considerations in Testing Semicustom and Custom ICs", Vol. 6, No. 3, pp. 20-42, March 1985
20. *91DVS, Device Verification Software, User's Manual* Manual # 070-6072-00, TEKTRONIX, INC., Beaverton, Oregon, September 1986

21. National Semiconductor, *TTL DATA BOOK*, National Semiconductor Corporation, 1976

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Chairman, Code 62 Department of Electrical and Computer Engineering Naval Postgraduate School Monterey, CA 93943-5004	1
4.	Curricular Officer, Code 32 Naval Postgraduate School Monterey, CA 93943-5004	1
5.	Professor C. Yang, Code 62Ya Naval Postgraduate School Monterey, CA 93943-5004	2
6.	Professor H. Loomis, Jr., Code 62Lm Naval Postgraduate School Monterey, CA 93943-5004	5
7.	Professor M. Cotton, Code 62Cc Naval Postgraduate School Monterey, CA 93943-5004	1
8.	Professor C. H. Lee, Code 62Le Naval Postgraduate School Monterey, CA 93943-5004	1
9.	Mr. David C. Schaeffer, Code 62 Naval Postgraduate School Monterey, CA 93943-5004	1
10.	Commander, Naval Research Laboratory ATTN: Mr. Andy Fox, Code 8120 4555 Overlook Ave., S.W. Washington, DC 20375	1
11.	Commander, Naval Research Laboratory ATTN: Lt. Brian Kosinski, Code 9110 4555 Overlook Ave., S.W. Washington, DC 20375	1

- | | | |
|-----|---|---|
| 12. | Dr. Allen Ross
Consultant
604 Lisa Lane
Bowie, MD 20715 | 1 |
| 13. | Mr. Howard Z. Bogert
Consultant
20720 4th Street, #12
Saratoga, CA 95192 | 1 |
| 14. | Professor Donald E. Kirk
Associate Dean of Engineering
School of Engineering
San Jose State University
San Jose, CA 95070 | 1 |
| 15. | Dr. Waldo G. Magnuson
Lawrence Livermore National Laboratory
L-156
PO Box 808
Livermore, CA 94550 | 1 |
| 16. | LCDR Walter F. Corliss, II
232 E. Rienstra Street
Chula Vista, CA 92010 | 2 |